Synaptic plasticity and memory addressing in biological and artificial neural networks

Danil Tyulmankov

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2024

Abstract

Biological brains are composed of neurons, interconnected by synapses to create large complex networks. Learning and memory occur, in large part, due to synaptic plasticity – modifications in the efficacy of information transmission through these synaptic connections. Artificial neural networks model these with neural "units" which communicate through synaptic weights. Models of learning and memory propose synaptic plasticity rules that describe and predict the weight modifications. An equally important but under-evaluated question is the selection of *which* synapses should be updated in response to a memory event. In this work, we attempt to separate the questions of synaptic plasticity from that of memory addressing.

Chapter 1 provides an overview of the problem of memory addressing and a summary of the solutions that have been considered in computational neuroscience and artificial intelligence, as well as those that may exist in biology. Chapter 2 presents in detail a solution to memory addressing and synaptic plasticity in the context of familiarity detection, suggesting strong feedforward weights and anti-Hebbian plasticity as the respective mechanisms. Chapter 3 proposes a model of recall, with storage performed by addressing through local third factors and neo-Hebbian plasticity, and retrieval by content-based addressing. In Chapter 4, we consider the problem of concurrent memory consolidation and memorization. Both storage and retrieval are performed by content-based addressing, but the plasticity rule itself is implemented by gradient descent, modulated according to whether an item should be stored in a distributed manner or memorized verbatim. However, the classical method for computing gradients in recurrent neural networks, backpropagation through time, is generally considered unbiological. In Chapter 5 we suggest a more realistic implementation

through an approximation of recurrent backpropagation.

Taken together, these results propose a number of potential mechanisms for memory storage and retrieval, each of which separates the mechanism of synaptic updating – plasticity – from that of synapse selection – addressing. Explicit studies of memory addressing may find applications not only in artificial intelligence but also in biology. In artificial networks, for example, selectively updating memories in large language models can help improve user privacy and security. In biological ones, understanding memory addressing can help with health outcomes and treating memory-based illnesses such as Alzheimers or PTSD.

Table of Contents

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my thesis advisor, Larry Abbott. I first encountered Larry during my undergrad, when he reviewed my first ever publication [1]. At that time, I had little idea of how important a role he would play in my career. Since then, the scientific skills that I learned will (hopefully) be evident in the rest of the dissertation, but the mentorship I received is harder to quantify. While some of my peers complained about their advisors, I was genuinely confused about their complaints: despite his fame, Larry was always available when I needed him, always present when I met with him, always provided support – both scientific and emotional – to help me grow, even letting me design my own course[1]. What I have come to admire most, however, is his unbounded curiosity and genuine love for the process of science. As I was told during my PhD interview, "Larry is a postdoc at heart". If I am fortunate enough to establish my own academic lab, I hope to emulate not only his success as a scientist, but more importantly, his wisdom as a mentor.

I am deeply grateful for the guidance and mentorship of Guangyu Robert Yang. As a postdoc at Columbia, Robert was instrumental in getting my PhD off the ground. He not only helped me assemble my first project from a collection of miscellaneous results into a single story, but also to understand its broader implications. Indeed, the unifying theme of this dissertation – memory addressing – is due to his suggestion. Always excited, enthusiastic, and supportive, he was the one to encourage me to aim high when publishing my first PhD paper (and it worked! [2]). With no topic off-limits for our conversations – from grand unifying theories of learning, to color schemes for posters – Robert was truly my guide into the deep end of research.

I am also most indebted to Dmitry Krotov. Despite my lack of physics chops, he agreed to take me on for an internship at the MIT-IBM Watson AI Lab, and I am lucky to have continued our collaboration to this day. To Dima's patience I owe my understanding of associative memory and energy-based models, which has fundamentally changed my thinking about both neuroscience and artificial intelligence in integral ways. Through Dima's mentorship, not only did my knowledge of

[1] NBHV GU4359, *Mathematical Tools for Theoretical Neuroscience*, Columbia University.

math and physics expand, but also my mastery of the Russian language, for the first time having a chance to discuss highly technical subjects in my mother tongue. Although his technical expertise matches the stereotype of the intimidating, hardcore, Russian theoretical physicist, I am grateful to know him as the kind, patient, humorous, and joyful mentor that he is.

Finally, I believe in the theory that neuroscientists study their own weaknesses. Mine is memory. Because I worry that I will forget an important name, at this point I will omit names entirely. Thank you to my family (Hi, Mom!). To my roommates, classmates, colleagues, and collaborators, many of whom became my friends. To my friends, some of whom became my colleagues and collaborators. I want to thank the directors and administrators who kept everything running (and to apologize to them for so often being delinquent in my PhD requirements). Thank you to the teachers and professors who taught amazing classes. Most importantly, thank you to all of my mentors, past and present – my thesis commitee; my undergraduate, Master's, and PhD rotation advisors; my internship managers; and everyone else who generously gave me their input and advice – all of whom inspired me and guided me on the quest that is scientific research.

Dedication

To educators everywhere.

Chapter 1: Memory addressing in biological and artificial neural networks

1.1 Introduction

It is widely accepted that learning and memory occur in brains through synaptic plasticity,[1] a biological mechanism for updating (potentiating or depressing) the efficacy of synaptic connections among a network of neurons. This process enables us to learn new skills – computations on external inputs performed by our brain which in turn actuate our bodies in useful ways – and acquire new memories – representations of previous experiences which inform our future behavior.

In computers, memory is distinct from computation. In neural networks, however, computation and memory are intertwined [5]. To produce a desired computation, a neural network must "learn" (i.e. program) it in the same way that a memory is "learned" (i.e. stored, written, or memorized) – through synaptic plasticity, although the timescales on which these are implemented can differ by orders of magnitude, ranging from seconds to generations. Indeed, because of the plastic nature of the brain, it may be difficult to draw the distinction between input stimuli providing operands for a computation or an algorithm, and stimuli modifying the operation itself [6]. Often, as in continual learning, the action of performing a computation leads to an update in the computation, so that next time it is performed slightly more efficiently, or using a slightly more accurate prior.

In this work, we define "memory" in a much more narrow sense, focusing specifically on storage and retrieval of singular items, as is the case for episodic or declarative memory. Familiarity or recognition memory (or, conversely, novelty detection) is a particularly useful testbed for investigating these mechanisms since it requires only reporting whether a particular item (e.g. an episode or piece of abstract knowledge) has been encountered. This makes readout much simpler than that for recall

[1] Although intrinsic neuronal mechanisms may also play a role in learning and memory [3]. In artificial neural network models, learning can even be done by updating only the gains and biases of neurons with a fixed set of synapses [4].

which requires reconstruction of the item, requiring less neural circuitry for decoding, thus making it easier to separate the memory from the computation. We (mostly) avoid the neuroscientific questions of synaptic or systems consolidation, as well as related questions from deep learning of function approximation, generalization, and prevention of overfitting. Indeed, we specifically focus on models which "overfit" – memorize the input data verbatim.

This implicitly suggests the hypothesis that although memory and computation can be intertwined, there may also be distinct neurons or brain areas dedicated for "memory" storage and retrieval (analogous to computer random-access memory, or RAM), which can operate based on some of the principles described here. The hippocampus, for example, is a plausible candidate for such a dedicated memory input-output device. Although we do not explore this idea further in this work, by symmetry, other neurons or areas may be dedicated for computational processing (analogous to a computer's central processing unit, CPU), the computations of which are defined (programmed) over extremely long-timescale mechanisms such as evolution.

For a memory device to be useful, it must implement the two aforementioned functions: storage and retrieval. In a biological or artificial neural network, *storage* corresponds to updating a set of synapses in response to an input stimulus, generally regarded as a pattern of activation in a subset of neurons, such that when the activation has decayed away, the process of *retrieval* can be used to reinstate the same pattern based on a cue, using the information that was stored by the synaptic updates. In other words, storage uses neuronal activity to drive updates in synapses. Retrieval uses the synaptic state to drive activity in neurons. Notably, storage and retrieval are separable. A failure to recall an item does not necessarily correspond its absence in memory storage. Rather, it may be a failure in retrieval. For example, it has been shown in a mouse model of Alzheimer's disease that retrieval of a memory that is inaccessible through natural recall cues is possible through optogenetic activation instead [7].

Most work on neural network mechanisms of memory discusses plasticity rules, explicitly seeking to establish a functional relationship between pre- and post-synaptic neuronal activity and a change in a synaptic weight that optimizes memory capacity [8], fidelity, or persistence [9].

The question of memory addressing for storage – which synapses should be the ones undergoing plasticity and how that selection is made – is often implicit or secondary in such studies. Addressing for memory retrieval poses a similar challenge – which synapses were responsible for storing an item, and how to access them to reinstate their information in neuronal activity. Although addressing and plasticity go hand-in-hand and are often interdependent, in this work we explicitly highlight the question of memory addressing during storage and retrieval, and attempt to (partially) orthogonalize it from the problem of synaptic plasticity. This is a useful perspective to help us reliably find and update, restore, or erase specific memories in a neural network without perturbing others. This may find applications, for example, in treating post-traumatic stress disorder (PTSD) or Alzheimer's disease in biological networks, or reducing privacy concerns due to memorization of personal information in artificial neural networks for artificial intelligence.

1.2 Memory addressing in computers

To set the context, we begin by discussing the more well-defined problem of memory addressing in computer systems [10]. Although many implementations of computer memory have been proposed [11], we will describe the most common one used today in personal computers – random-access memory (RAM). This type of memory is called "random-access" because data can be read from or written to any physical location on the storage device, in any order, in the same amount of time.

RAM is comprised of units called *memory cells*. Each memory cell stores one bit of information, either a 0 or a 1. Memory cells are arranged into a *memory array*, with each of 2^N memory cells corresponding to an N-bit address. To access the information stored in a memory cell, the processor sends a memory address along N *address lines* (one for each bit in the address) to an *address decoder* – a circuit which converts the N-bit address to a 2^N-bit representation where exactly one of the 2^N bits is active. This "one-hot" representation selects the corresponding memory cell in the array, the contents (one stored bit) of which are then sent back to the processor via a *data line*. For example, an address of length $N = 3$ can identify $2^n = 8$ different memory cells, and the address "`011`" would correspond to the 4^{th} cell. To write data, the memory cell is selected in the same way,

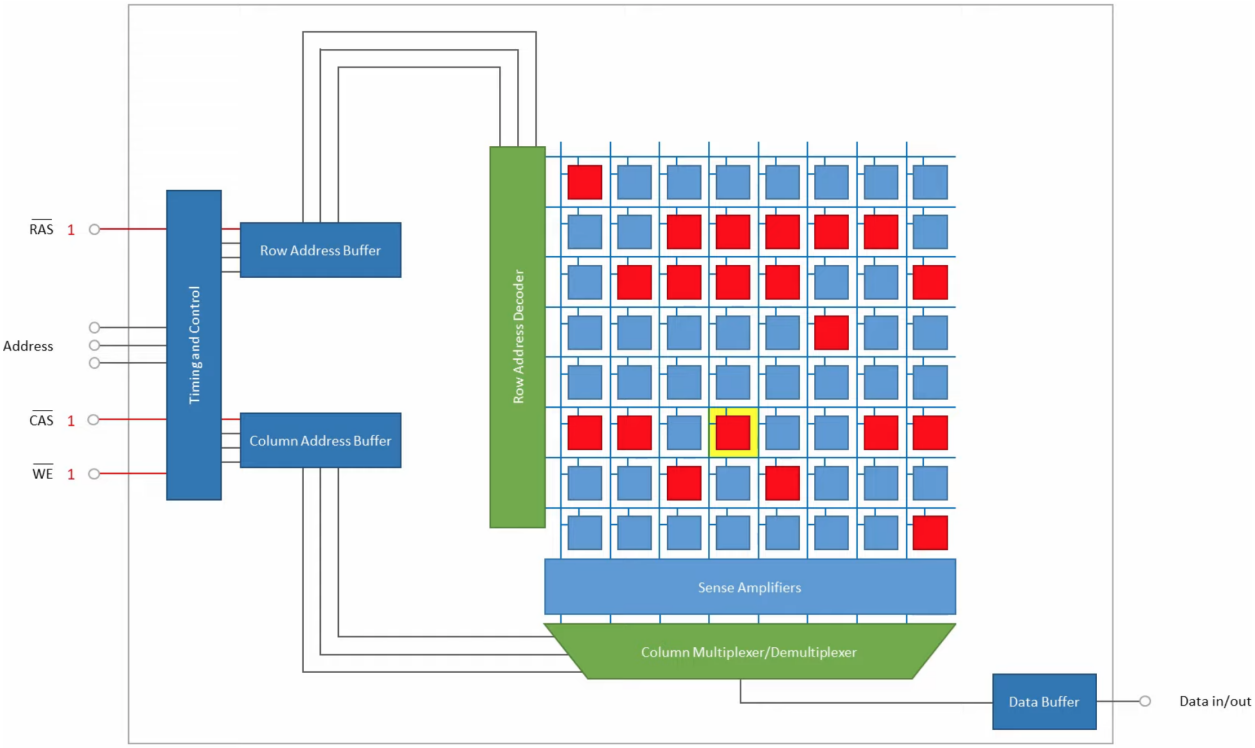


Figure 1.1: Schematic of a DRAM array with 64 memory cells arranged in 8 rows and 8 columns, with $m = k = 3$ address lines. Image from [12].

but data is received from the processor along the data line. An auxiliary input controls whether the memory array should be in read or write mode.

In practice, a memory cell in RAM (specifically, dynamic RAM or DRAM) consists of a transistor and a capacitor, where the information bit is encoded as a high (1) or low (0) charge on the capacitor, and the transistor gates current flow from/to the capacitor for reading/writing. The memory cells are arranged in 2^m rows and 2^k columns (where $m + k = N$) to make the memory array (Figure 1.1). To access a memory cell, the device operates in two steps. First, it receives an m-bit string *row address* from the processor, which is sent to the *row address decoder*. As before, the decoder converts this to a one-hot representation and selects the corresponding row by opening the gates of all transistors in that row, thus letting current flow from the capacitors. All 2^k bits of data in this row are read out by 2^k circuits called *sense amplifiers*.

Next, the device receives a k-bit *column address*, which is sent to a *column multiplexer* along with the 2^k data bits from the sense amplifiers. The multiplexer acts as a 2^k-way switch to select

the input corresponding to the k-bit column address, similar to the row address decoder, which is then sent out through the data line to the processor. The rationale behind this grid layout is to reduce the number of address lines, requiring $\max(m, k)$ instead of N lines, thus simplifying the problem of routing wires on a circuit board and minimizing the number of pins on the memory device. In practice, the column multiplexer and sense amplifiers take up much more space than the row decoder, so $m > k$, requiring m address lines.

This process reads/writes a single bit of information from/to memory. To read/write B bits in parallel, for example a "byte" ($B = 8$ bits) or a "nibble" (half a byte, $B = 4$ bits), we can replicate this memory array B times to create a *memory bank*. Each array in the bank receives the same address, but has its own data line. As a result, when the processor selects one N-bit memory address, it receives (or is able to write) B bits along B data lines. This entire process is extremely fast, occurring on the order of nanoseconds.

1.3 Synaptic plasticity

While a computer might store a bit of information by charging or discharging a capacitor, a neural network stores information by changing the strength of its synaptic connections (but see [5] for a review of other neural mechanisms of memory). A comprehensive review of synaptic plasticity is beyond the scope of this work, but we provide an overview to set the context for the complementary problem of memory addressing discussed in the remainder of this chapter.

A plethora of biochemical processes regulate the strength of a synapse, which can be measured as the initial slope of a post-synaptic potential in response to a pre-synaptic spike [13], or the amplitude of a post-synaptic current [14]. From a computational perspective, however, synaptic plasticity commonly refers simply to the update of a scalar "synaptic weight". A post-synaptic neuron's activity (modeled either explicitly as a series of action potentials, or abstracted into a firing rate) is controlled by its membrane potential, which is set by the total input current into the cell – a weighted sum of all the currents transmitted by its pre-synaptic partner neurons. The weighting in this sum is determined by the synaptic weights of the synapses connecting each pair of neurons.

Although, in general, synaptic plasticity may refer to any such weight modification – including random fluctuations [15] or homeostatic mechanisms [16] – as it is commonly studied, plasticity is the result of experience-driven updates that induce "learning" or "memory" in an organism or a model.

In a model, the implementation for computing the synaptic weight update can be biological or not. The most famous and foundational proposal for a biological synaptic weight update rule is Hebb's postulate [17], "neurons that fire together, wire together." In other words, if the firing rates between two neurons are sufficiently correlated, the synapse between them will be potentiated (strengthened). Mathematically, in its simplest form, the update Δw_{ij} for the synaptic weight w_{ij} between neurons i and j, with firing rates x_i and x_j can be expressed as simply the product of the firing rates,

$$\Delta w_{ij} = x_i x_j$$

If either neuron is silent, there is no update. If they are co-activated, the synapse is potentiated. More generally, a "Hebbian" rule has become known to refer to any function of the pre- and post-synaptic activity, and potentially the synaptic weight itself,

$$\Delta w_{ij} = F(w_{ij}, x_i, x_j)$$

Numerous models of plasticity building on the Hebbian rule have been proposed. For example, Oja's rule [18] ensures its stability and implements unsupervised learning (principal components analysis, PCA). The BCM rule [19] enables bidirectional plasticity – not only potentiation but also depression – with a sliding threshold to determine the sign of the update. In Chapter 2 of this thesis, we consider "anti-Hebbian" plasticity, where neurons that fire together wire *apart* (correlated neural activity causing synaptic depression) as a mechanism for familiarity detection [2]. In Chapter 3, we consider a "neoHebbian" [20] three-factor plasticity rule for memory recall, which incorporates not only pre- and post-synaptic activity but also a modulating factor [21].

More biophysically detailed models which implement individual action potentials of a neuron

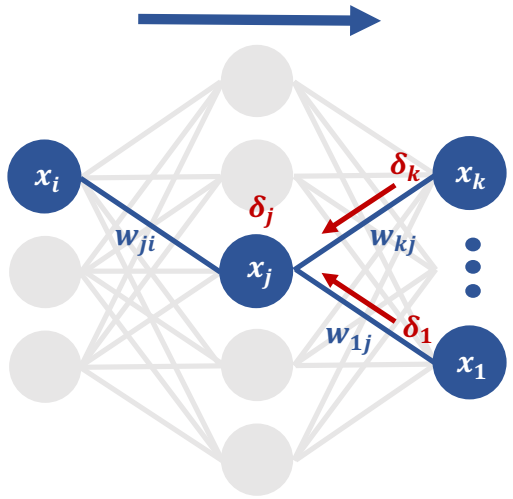


Figure 1.2: Backpropagation in a feedforward network. Errors δ are backpropagated (red arrows) through the network through weights w_{jk} symmetric to the feedforward weights w_{kj} (forward information propagation indicated by green arrow), causing a biological implausibility known as the weight transport problem. Image adapted from [26].

(rather than abstracting them into a scalar firing rate) consider plasticity rules that depend on the temporal pattern of action potentials such as spike-timing dependent plasticity [22] or triplet rules [23]. Including individual ions in the model suggests plasticity such as calcium-based rules [24]. Considering more complex circuit dynamics such as bursting leads to history-dependent structural plasticity [25] . All of these can also be considered "Hebbian" plasticity rules as they only involve information from pre- and post-synaptic neurons.

In modern "deep" artificial neural networks, synaptic weight updates are computed by the backpropagation of errors algorithm [27], generally considered biologically implausible. The synaptic weight update is the product of pre-synaptic activity and the post-synaptic neuron's error. This error is computed recursively as the weighted sum of errors from neurons downstream of the post-synaptic neuron, weighted by the post-synaptic neuron's efferent synaptic weights. As implied by the name, the errors are propagated backwards from downstream neurons (closer to the output layer) to upstream (closer to the input) through weights that are symmetric to the network's feedforward weights (Figure 1.2). This symmetry, or "weight transport" problem [28, 29] is difficult to reconcile biologically, although alternative biologically plausible solutions have been proposed

[30, 31].

In a recurrent neural network (RNN), the problem of computing synaptic weight updates to achieve the desired dynamics is more complicated. A synapse needs to consider its effect on the network's input/output function not only with respect to the other synapses, but also to the entire trajectory of the neural dynamics. In an artificial neural network, this is solved by extending the backpropagation algorithm to "backpropagation through time" (BPTT), which considers the network's history by "unrolling" the recurrent computation. Each timestep is considered a layer of a deep feedforward network with identical weights between layers, and the backpropagation algorithm is applied. However, not only does it suffer from the weight transport problem, but it also requires neurons to store their history of activity [32]. Various biological solutions to this problem have been proposed as well [33, 34, 35, 36].

In the special case of RNNs that obey dynamics which converge to a fixed point, a simpler algorithm known as recurrent backpropagation [37, 38] can be used, taking advantage of the fact that once the neural dynamics have converged to a fixed point, the trajectory by which they arrived is irrelevant (see section 7.2 of [39] for a more didactic presentation and [40, 41] for a modern view). In Chapter 5 of this thesis, we evaluate this algorithm in the context of the modern Hopfield network [42] – a fixed-point RNN which is amenable to analysis from a memory addressing perspective (see Section 1.4.3 and Section 1.4.6) – and connect it to Hebbian plasticity.

Whether expanding a network's storage capacity [43], enabling continual learning [44, 45], or inferring the plasticity rules [46, 47], much of the theoretical work on learning and memory focuses on improved mathematical formulations of "synaptic plasticity". Here, we turn to a less well-studied component of this problem, "memory addressing". Namely, the selection of synapses which undergo plasticity, and the counterpart, their selection for subsequent retrieval.

1.4 Memory addressing in neural networks

1.4.1 Ideal observer models

We begin by describing a synaptic modeling paradigm that avoids the problem of memory addressing altogether. In the "ideal observer" approach [9], we consider a set of synapses, represented as a collection of weights W, without regard to the neurons which they connect or network architecture they are part of. A "memory" in this case is simply the incremental change (potentiation or depression) Δw in the set of synaptic weights. Subsequent synaptic updates occur randomly through ongoing activity, e.g. due to storage of memories, or random synaptic fluctuations [43, 48, 49], although in general they can also be determined by a specific learning paradigm (and therefore specific memory addressing mechanism) such as supervised, unsupervised, or reinforcement learning [50].

To establish whether a memory remains available for recall, we consider its signal-to-noise ratio (SNR). The signal corresponds to the overlap between the memory Δw and the current synaptic state, the exact definition of which depend on the specifics of the values for weights and statistics of the updates [51]. The noise is the standard deviation of the signal. We can then quantify the lifetime of the memory by calculating the number of subsequent synaptic updates it takes for the SNR to drop below an arbitrary threshold (although the value depends on the threshold, the scaling of the temporal lifetime does not). This approach avoids the need to design a mechanism from extracting the information stored in synapses, simply assuming that the brain is capable of operating optimally to access this data. As a result, this is an upper bound on the memory capacity, with realistic readouts likely performing sub-optimally.

The ideal observer setup can also be thought of as a trivial (but foundational [52] and well-studied [53]) neural network architecture with N input neurons connected through N synapses to a single output neuron (Figure 1.3), performing a recognition task [54, 55]. The output neuron's goal is to report whether a stimulus has been previously presented, with its activity computed as the dot product between the input vector and the synaptic weights, i.e. their overlap, and SNR can be

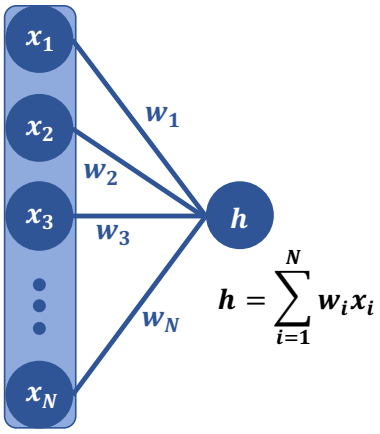


Figure 1.3: Single neuron, equivalent to an "ideal observer" setup.

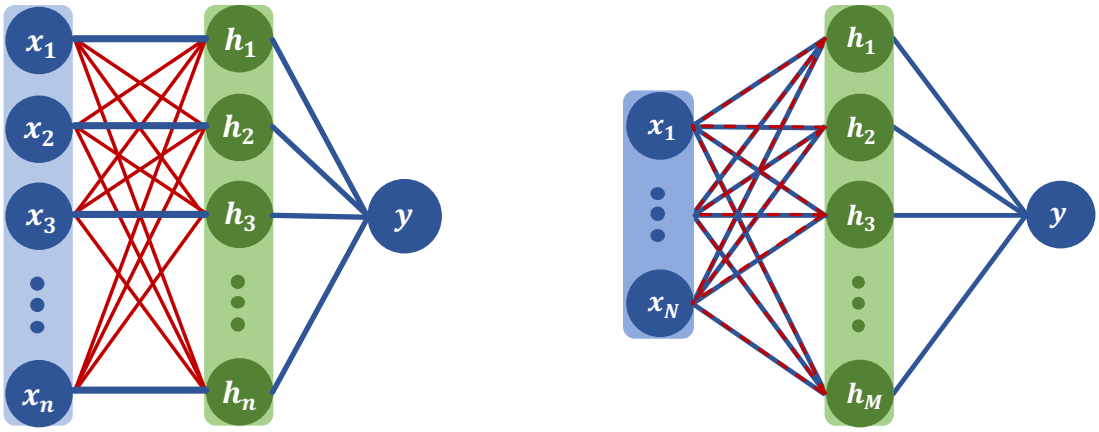
computed from this value.

The problem is significantly more complicated when the synapses are arranged in even a simple topology – one hidden layer (Figures 1.4 and 1.5). Now, storage/readout of information requires carefully selecting the synapses to update/query. For storage, synaptic plasticity rules which are typically considered "biologically plausible" must be local to the synapse, i.e. only depend on the activity of pre- and post-synaptic neurons (although see Section 1.4.5). Since multiple synapses project to a single neuron, synapses are no longer independent and can no longer be selected arbitrarily as in the ideal observer model. Furthermore, unlike the topology in Figure 1.3, input patterns are not necessarily one-to-one mapped to elements of the synaptic update vector Δw. In the rest of this work, we discuss potential solutions to this problem.

1.4.2 Strong feedforward weights

Feedforward weights for recognition

Extending the simple topology in Figure 1.3 of a single recognition neuron, we can consider one with multiple recognition neurons, all of which receive the same input (Figure 1.4). The familiarity signal is now encoded in their population activity, so an additional output neuron y is introduced to decode this population signal. If the hidden neurons are identical, however, their population activity will be equivalent to that of a single neuron.



(a) (b)

Figure 1.4: Networks with one hidden (addressing) layer for familiarity detection. Blue lines indicate fixed synaptic weights used for addressing. Red lines indicate plastic weights used for memory storage. (a) One-to-one strong feedforward weights ensure the address is equal to the input. Adapted from [56]. (b) Synaptic weights for addressing overlap with those used for storage. Meta-learned fixed weights enable having an arbitrary address for each input as well as decoupling of the input and hidden layer activities. Adapted from [2].

As one solution to this problem, [56] propose breaking the symmetry in the hidden neuron activities with strong feedforward weights. Considering a network with N input units and N hidden units, the authors propose strong synapses from the i^{th} input neuron to the i^{th} hidden neuron (Figure 1.4a). These weights are fixed, while the remaining ones encode the (binary) input stimulus according to a Hebbian plasticity rule. During both storage and readout, this mechanism ensures that the only hidden units which participate in the memory are those whose corresponding input unit is active. The input serves not only as the stimulus to store, but also as the address at which to store it.

A similar solution emerges through optimization ("meta-learning") of neural network parameters for a continual familiarity task [2]. Unlike the one-to-one feedforward weights proposed in [56], however, the fixed feedforward weights use a subset of input neurons to select the hidden neurons which participate in the memory storage and readout (Figure 1.4b), resembling the sparse version of the sparse distributed memory (SDM) address decoder matrix [57] discussed below. This not only decouples the size of the input layer from that of the hidden layer, enabling the expansion of the storage capacity without increasing the dimensionality of the input, but also increases sparsity in the hidden layer activity (one-hot in the idealized case), therefore further decoupling the hidden neurons

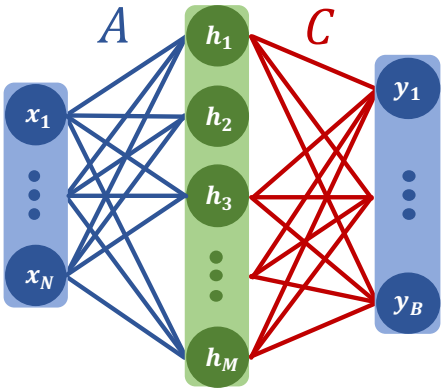


Figure 1.5: Sparse Distributed Memory network with one hidden layer for recall. Fixed addressing synaptic weight matrix A selects a hidden unit (the "address") according to the input x, which indexes into the plastic content matrix C and retrieves the corresponding memory y. Adapted from [60].

from each other. This work is discussed in detail in Chapter 2 of this thesis.

Feedforward weights for recall

A simple network for storing and recalling items, originally proposed as a model of human long-term memory by [58] can function analogously to RAM, using feedforward weights to address stored items [59]. In its simplest instantiation, the sparse distributed memory (SDM) is a three-layer fully connected neural network (Figure 1.5) with binary inputs, outputs, and synaptic weights. Matching the notation in section 1.2, the network is comprised of an input layer of size N, a hidden layer of size $M = 2^N$, and an output layer of size B. The input-to-hidden synaptic weight matrix A is fixed and its rows enumerate all the 2^N length-N binary strings. Now, given any binary input vector, exactly one row of A will match the input. Therefore, using a step function nonlinearity in the hidden layer with an appropriately set threshold, we can ensure that exactly one hidden unit is active. This layer functions like the RAM address decoder, converting an N-bit address to a length-2^N one-hot representation in the hidden layer.

Propagating this one-hot hidden layer activity forward through the hidden-to-output synapses, which we call the content matrix C, we index into exactly one (length-B) column of this synaptic weight matrix. This corresponds to the B synapses efferent from the active hidden neuron. Each

neuron in the output layer will receive an input proportional to the single synaptic weight projecting from the active hidden neuron, and therefore the contents of those synapses will be set in the activations in the output layer and available for readout. To update the contents of that slot, we select the hidden layer neuron the same way and, clamping the output layer to the desired value, update the synapses according to a Hebbian plasticity rule: synapses afferent onto silent neurons get depressed and those afferent onto active neurons get potentiated (and, in both cases, thresholded to remain in the range $\{0, 1\}$).

A potential problem with this design is the exponential size of the hidden layer. One way to ameliorate it is by taking only a subset of addresses (rows) in A, e.g. $M << 2^N$, and setting the hidden layer threshold such that a small number (k) of hidden units are active in response to an input (hence, "sparse"), rather than exactly one. These hidden units correspond to the k addresses closest in Hamming distance to the input. Now, when storing an item in C, it will be stored in a "distributed" fashion in k slots (columns of C), and the retrieved pattern will be their average, introducing noise in the recall process. Nevertheless, the memory is accessed through the fixed addressing matrix A, although the address now maps onto a distributed (k-hot) rather than localized (one-hot) representation in the hidden layer.

Other extensions of the addressing mechanism of SDM have been considered as well. A sparse addressing mechanism was proposed by [57, 61], where the addressing matrix only considers a subset q of the N input bits and activates the corresponding hidden unit if all q bits match those in the address (this can be thought of as the binary input being represented by a vector in $\{+1, -1\}^N$, rather than $\{0, 1\}^N$, and each row of A having q entries set to $+1$ or -1, and the rest set to zero).

1.4.3 Content-based addressing

So far, we have discussed location-based addressing in which an input is used to query a particular address, defined by a fixed feedforward synaptic weight matrix, from which the stored value is returned. Addressing with feedforward weights compares the input and a set of stored addresses, selecting the one which has the largest overlap with the input. The value at that address can be

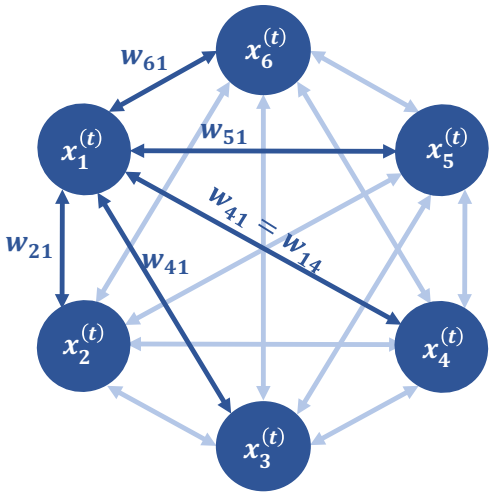


Figure 1.6: Classical Hopfield network with $N = 6$ neurons, recurrently connected through a symmetric synaptic weight matrix W.

updated or replaced. An alternative memory system is one which can be queried using its stored contents rather than an address. Content-addressable memory (CAM), also known as associative memory, stores items which can be retrieved by presenting a perturbed version of the item as the input (e.g. partially blanked, bits flipped, or noise added), and will retrieve the original item as the output. Content-based addressing has been considered as a memory paradigm for computer systems for many decades [62] through to recent years [63], various neural network implementations of which have been proposed as models of biological memory.

Implicit content addressing

The classical example of a neural network implementation of associative memory is the Hopfield network [64]. A Hopfield network consists of N neurons fully interconnected by N^2 synapses (Figure 1.6) and stores P items $\{\xi_1, \ldots, \xi_P\}$, where each ξ_i is an N-dimensional binary vector in $\{+1, -1\}^N$. To store item ξ_i, it is represented as a synaptic update given by a Hebbian plasticity rule,

$$\Delta W = \frac{1}{P}\xi_i\xi_i^\top$$

(note ΔW is an $N \times N$ matrix, but could also be reshaped and thought of as a length-N^2 vector Δw, as described in Section 1.4.1). Readout of an item proceeds according to the dynamics

$$x^{(t+1)} = \text{sign}(Wx^{(t)})$$

where $x^{(t)}$ is an N-dimensional binary vector representing the state of the neurons at time t. The initial state $x^{(0)}$ is a perturbed version of a stored item, and the retrieved item is the resultant fixed point x^* of these dynamics, i.e. $x^* = \text{sign}(Wx^*)$. Many extensions of this network have been considered, including continuous versions [65], hetero-associative memories [66], or those with extremely large storage capacities [67, 68, 69].

The difficulty with such models is that the memory representation ΔW stored in the synaptic weights is fully distributed and overlapping with other stored representations. This makes it challenging to isolate an individual memory the way that we could with, for example, the SDM, where a single fixed bit string indexes a row (or a few rows) of the contents matrix, allowing direct access to a single memory stored in the synaptic weights. In other words, in the classical Hopfield network, memories do not have an explicit "address." There are, however, generalizations of this model reminiscent of the SDM which use modifiable content-based addresses (in combination with various content selection mechanisms – see Section 1.4.4) rather than addressed locations to store and retrieve items. We will now discuss such models.

Explicit content addressing

The Modern Hopfield Network (MHN) [69] or Dense Associative Memory (DAM) [68] can be generically implemented as a neural network with two layers of recurrently connected neurons (Figure 1.7) [42]: a visible layer which projects to a hidden layer through synaptic weight matrix W, which recurrently projects back to the input layer through symmetric synaptic weights $W^\top$. The choice of nonlinearity in the hidden layer reduces this general setup to specific cases of this network, such as the DAM for a polynomial nonlinearity, or the MHN for a softmax function in the hidden layer.

The rows of the input-to-hidden synaptic weight matrix W (equivalently the columns of the hidden-to-input weights $W^\top$) are the memory slots, analogous to those of the content matrix C in

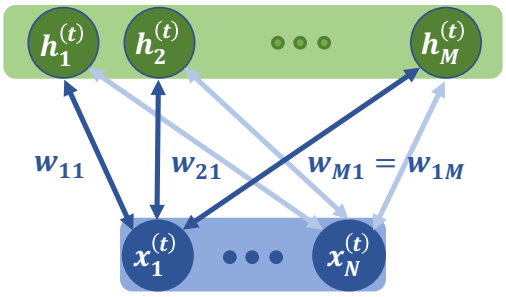
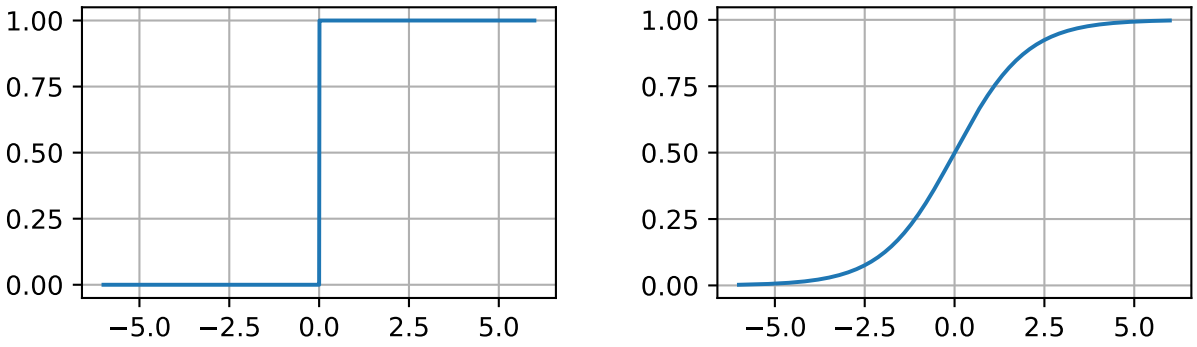


Figure 1.7: Modern Hopfield network, or Dense Associative Memory. A visible layer x projects to a hidden layer h, which projects back to the visible layer through symmetric synaptic weights. The hidden layer may be interpreted as an address, analogous to Figure 1.4 or Figure 1.5. Adapted from [70].

the SDM. Now, however, the contents also serve as the addressing mechanism for lookup – to locate an item, rather than using the overlap between the input and a fixed address, we consider the overlap between the input and the stored contents. Upon the initial presentation of a perturbed input, a subset of hidden units are activated, each unit's membrane potential proportional to the overlap (dot product) between the input and the vector of synaptic weights afferent onto that unit. Their activity is further sparsified by the activation function (discussed in Section 1.4.4). The hidden activity is then projected back to the input layer through $W^\top$, the membrane potential of which is the weighted sum of the stored memories (columns of $W^\top$, or equivalently rows of W), weighted proportional to the hidden unit activities. These recurrent dynamics continue until convergence to a fixed point, where the activity of the input layer has converged to a retrieved item, addressed by one or more hidden units, which index corresponding memories stored in the synaptic weights (rows of W).

Note that the classical Hopfield network is also a special case of this model, with the activation function being the identity for the hidden layer and the $\mathrm{sign}(\cdot)$ function for the input layer. In this case, the effective weight matrix is the outer product of the explicit weight matrices that project from the input to the hidden layer and back, $W_{\text{eff}} = WW^\top$. In this way, the classical Hopfield network can be thought of as a MHN with the addressing mechanism (i.e. the hidden layer activity) integrated out.

A similar model is used in [21], although without weight symmetry and without recurrent dynamics. We consider a network with the same topology as the SDM – an input, hidden, and output



(a) Heaviside step function

(b) Logistic function

Figure 1.8

layer. The input-to-hidden synapses comprise the "key" matrix, analogous to the addressing matrix of the SDM. The hidden-to-output synapses store the "value" analogous to the contents matrix. Memory retrieval proceeds in a feedforward manner, with the key serving as a memory address. Unlike the SDM, however, items are stored as key-value pairs, so the addressing (key) matrix is updated dynamically alongside the contents (value) matrix. For this reason, we refer to this model as a key-value memory (KVM). Note that it can also be used as an auto-associative memory like the classical Hopfield network by setting the key equal to the value. The mechanism for writing memories is discussed in Section 1.4.5.

1.4.4 Content selection mechanisms

Both in the case of location-based addressing implemented by strong feedforward weights, as well as that of content-based addressing where memory slots are selected based on their similarity to the cue, the hidden layer activity undergoes further refinement to achieve a desired level of sparsity. This level of sparsity defines the number of slots activated during retrieval of a memory. In this section, we discuss some of the mechanisms by which the hidden layer activity can perform this computation.

Thresholding

The Heaviside step function

$$\Theta(x) = \begin{cases} 0 \text{ if } x < 0, \\ 1 \text{ if } x \geq 0 \end{cases}$$

(Figure 1.8a) is a common activation function to use for neuron models which require a binary decision, including those for pattern recognition [71], spiking neural networks [72], or, as described in Section 1.4.2, address selection.

The sparse distributed memory (SDM) [73] (Section 1.4.2) uses the step function $\Theta(x - b)$ in the hidden layer to ensure that exactly k hidden units ($k = 1$ in the special case where SDM behaves like RAM) are activated in response to an input corresponding to an address. The threshold b must be carefully adjusted based on prior knowledge of the input currents into the hidden layer such that exactly the neurons above this threshold are activated (output a "1") and all others are silent (output "0"). In a different example, in a familiarity detection network model, rather than a fixed threshold, the authors [74] use a dynamic threshold to ensure that the coding level of the hidden layer (i.e. fraction of hidden units active) is independent of the coding level of the input. The threshold is proportional to the number of active input neurons, implemented with an inhibitory neuron that receives projections from the input neurons and projects to the hidden layer.

Alternatively, the logistic function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

(also called the sigmoid function[2]) can be considered a "soft" version of the step function, where the output grows smoothly from 0 to 1 as the input increases (Figure 1.8b). From a modeling perspective, the primary advantage of the logistic function as a neuronal nonlinearity is its differentiability. Unlike the step function, which has derivative equal to zero everywhere that it's defined, the logistic

[2]This common but imprecise, as any "S-shaped" function is a sigmoid, including the logistic function, but also functions such as $\tanh(\cdot)$, $\arctan(\cdot)$, or $\text{erf}(\cdot)$.

function is differentiable everywhere. As a result, gradients can backpropagate through it, enabling learning (or meta-learning [2]) in the network with a gradient descent based algorithm. Additionally, unlike the step function, the logistic function has a tunable gain (i.e. slope of the function at the origin), making the scale of the input meaningful, unlike the step function where $\Theta(ax) = \Theta(x)$ for any constant a. As the gain increases to infinity, the logistic function approaches the step function.

Recurrent inhibition

One limitation of using a neuron-wise threshold function (or soft-threshold in the case of the sigmoid) to control the sparsity of the hidden layer activity is the need to fine-tune the threshold value based on the statistics of the inputs into each neuron. If these change over time, the threshold needs to be re-adjusted. Alternatively, it is possible to simply select the unit or units with the highest membrane potential (controlled by the sum of input currents) and only have those fire, while the rest remain silent. This function is known as the winner-take-all (WTA), or more generally the k-winners-take-all (k-WTA) function, where the top k elements of an n-dimensional input are returned as 1 and the rest are 0.

The price for this functionality, however, is additional anatomical complexity. To implement k-WTA, we require recurrent (lateral) inhibitory connections among the neurons participating in the function [75, 76]. Neurons with the highest firing rates send the strongest inhibitory inputs to the rest of the circuit, further suppressing their activity and eventually silencing them completely. Although such a circuit implementation can be explicitly modelled for a memory task [77], it is common to implicitly assume that it exists and simply consider the hidden layer collective "nonlinearity" to be a function (vector field) from $\mathbb{R}^N$ to $\{0, 1\}^N$ (rather than element-wise) which selects the k largest inputs (with $k < N$). This has been used for addressing both in models of familiarity [56] and recall [78].

The continuous analogue of the k-WTA function is the softmax function, given by

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

Rather than setting the activity of the most-active units to a high value and the rest to zero, the softmax performs normalized exponentiation such that units with high inputs are enhanced and those with low inputs are suppressed. The entire vector is normalized such that it sums to 1, thus allowing it to be interpreted as a probability distribution. Like the k-WTA function, it is possible to implement the softmax with inhibitory recurrent connections and explicit dynamics [79], although in practice it is treated as a collective nonlinearity applied to the entire input vector.

The softmax function is classically used as a generalization of the logistic function in the output layer of a multi-class classifier network, although in recent machine learning literature it has been used as an "attention" mechanism [80], most notably in the Transformer architecture [81]. Although it is not performing recall of stored memories as described above, the softmax attention is being used by the Transformer decoder in a similar way to address into the input sequence to select the most relevant items for generating the next output token. Indeed, the attention mechanism of Transformers is equivalent to the update dynamics of the modern Hopfield network [69]. This is made explicit in the neural network description of the MHN, using a softmax nonlinearity in the hidden layer [42]. The same mechanism is used for memory retrieval in the key-value memory network from [21], described in Chapter 3 of this thesis.

Like the logistic function, the softmax also has a tunable gain. Following terminology from statistical mechanics, this is known as the inverse-temperature parameter, commonly denoted as β which controls the "peaked-ness" of the function. As β approaches infinity, the softmax approaches a one-hot vector; as β approaches zero, the softmax approaches a uniform distribution. In the context of memory addressing, this parameter can be used to control the specificity of the address – a one-hot address corresponds to the contents of exactly one slot, or a distributed address returns the weighted average of several slots. This, in turn, impacts the type of stored representation – a "prototype" representation where a single slot represents an individual item, or a "feature" representation where multiple slots must be combined to output a stored item from memory. This is discussed further in Chapter 4.

1.4.5 Three-factor plasticity

Thus far, in most of the examples of memory (recall or recognition) networks we have discussed, storage of input stimuli has been performed through a Hebbian-like plasticity rule, with various mechanisms in place to select the pre- and post-synaptic neurons which will be co-activated and their corresponding synapse potentiated (or depressed). Both theoretical and experimental evidence, however, shows that "third factors" such as reward, neuromodulators, dendritic plateau potentials, or signaling molecules internal to the neuron/synapse can play a role in driving plasticity, or modulating its timing, magnitude, or even its sign. Although three-factor plasticity has been used in a variety of scenarios [82], here, we consider the perspective of the third factor as the addressing mechanism for selecting neurons (and therefore their synapses) to be updated for memory storage.

Global third factors

Three-factor plasticity rules [83] classically rely on "global" third factors such as reward, novelty, or attention, corresponding to dopamine, acetylcholine, noradrenaline, or other neuromodulators which impinge globally on all plasticity sites in a learning circuit to control plasticity. Mathematically, it is represented as a scalar value common across all synapses which modulates a Hebbian (two-factor, i.e. pre- and post-synaptic activity) plasticity rule.

The classical example of a global third factor comes from reinforcement learning [84], in which a population of cells s representing spatial location projects to another population a representing actions. If the action performed at a given location, given by $a = Ws$, leads to a reward, all the synapses encoding that action are strengthened in proportion to the reward value r (or, more precisely, the deviation from the expected reward),

$$\Delta w_{ij} = r a_i s_j$$

In a more recent example from supervised learning [85], a recurrent neural network is endowed with ongoing synaptic plasticity which follows a Hebbian rule. One of the network outputs corresponds

to a time-varying third factor $M(t)$ referred to as "backpropamine", which controls the scale and timing of the plasticity updates across all synapses,

$$\Delta w_{ij} = M^{(t)} h_i^{(t)} h_j^{(t)}$$

During training, this network must learn to generate its own global third factor for gating plasticity. A global third factor is also used in [21] (see Chapter 3) to switch between storage and retrieval phases of a recall task.

Local third factors

Due to its scalar nature, a global third factor can be regarded as a mechanism for gating "whether" or "when" rather than "where" plasticity occurs, which is the focus of this chapter. "Local" third factors, on the other hand, have the spatial specificity to control plasticity in individual neurons or synapses [20]. A compelling piece of experimental evidence for local third factors comes from [86], where the authors observe that the formation of a hippocampal place cell occurs only as a result of a strong prolonged depolarization in the apical dendrites of pyramidal neurons, known as a dendritic plateau potential or a calcium spike.

In the most general case, appropriately controlled local third factors can have been theorized to solve the credit assignment problem in a biologically plausible manner [30] (see Section 1.4.6). Here, we suggest a simpler alternative for the role of local third factors (e.g. dendritic plateau potentials). Consider either the modern Hopfield network [42] or similarly the key-value memory [21]. Rather than encoding an error signal as suggested by [30], a local third factor can simply activate one or several hidden neurons, selecting their corresponding afferent, efferent, or both sets of synapses for writing. Note that in the MHN, afferent synaptic weights are equal to efferent synaptic weights due to weight symmetry, but this is not the case for the KVM.

Local third factors may be triggered randomly, sequentially [21], based on recency of activation [87], or in another systematic ordering. A random activation can be performed through intrinsic

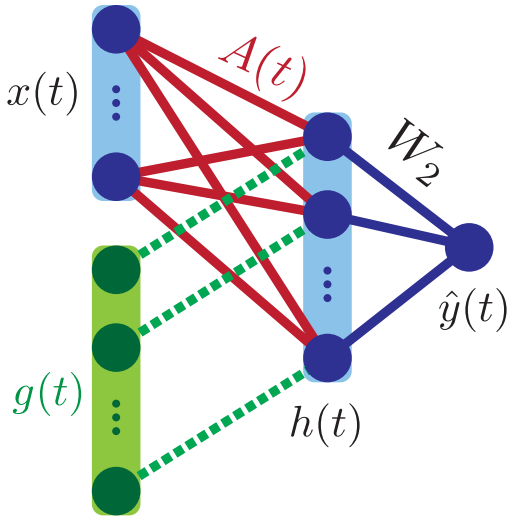


Figure 1.9: Local third factors generated by an external gating circuit (green) define the addressing function by selecting hidden neurons for memory storage in a familiarity detection task. Image from [89].

neuronal mechanisms, for example where noise-induced membrane potential fluctuations cross a threshold to generate a plateau potential. Sequential activation can be achieved through timing mechanisms, where each of N neurons generates a local third factor approximately every $\frac{1}{N}$ seconds, for example by integrating a tonic input until reaching a threshold and resetting. Synfire chains [88] are another potential mechanism for generating a sequence, where one neuron's local third factor triggers activation of the next one through feedforward connectivity. An integration mechanism can also be used to determine the most- or least-recently-used neuron by integrating the history of local third factors and comparing this value across neurons using a WTA function. Finally, control of the ordering can be done by an external gating circuit [89] which generates a control sequence in any of the aforementioned ways and triggers local third factors through dedicated projections to the addressing (hidden) neurons (Figure 1.9).

Selection of hidden neurons through local third factors is analogous to the way in which strong feedforward weights in the sparse distributed memory [59] select a subset of hidden neurons through an explicit addressing matrix, whose efferent synapses (i.e. columns of the content matrix) are updated with a Hebbian rule (see Section 1.4.2). Unlike the SDM, however, when storing an item, the address selection with local third factors is independent of the existing synaptic weights, whether

they correspond to fixed addresses or to "keys" of previously stored items. Moreover, rather than a standard Hebbian rule involving pre- and post-synaptic activity, the "neo-Hebbian" [20] plasticity rule now involves a local third factor.

1.4.6 Credit assignment

When considering a topology that is more complicated than a set of synapses afferent onto a single neuron, addressing mechanisms are required to select the synapses to update for storage, or to query for retrieval. We have discussed several solutions for the simple neural architecture of one hidden layer. The problem becomes less tractable when considering truly "deep" neural networks with many hidden layers (Figure 1.10) or more complicated architectures such as residual or convolutional neural networks [90], recurrent neural networks [91], or Transformers [81]. Rather than an explicit mechanistic algorithm for storing items, such networks are typically trained with gradient descent methods [92] to perform tasks such as classification, regression, translation, generation, or decision making. In this case, the weight update,

$$\Delta w = -\eta \nabla_w L$$

does not correspond to storing an additional item in memory, but rather to an incremental correction to the network to improve its task performance as measured by the loss function L, with the size of the increment controlled by the learning rate η.

The problem of computing the synaptic weight updates is referred to as "credit assignment", because each synapse needs to be given credit (or, conversely, blame) for its contribution to the loss, while factoring in contributions of all the other synapses in the network. This is classically solved by the backpropagation of errors algorithm [27], although it is generally considered biologically implausible. More realistic alternatives have been proposed [93, 94] performing credit assignment by a three factor rule (see Section 1.4.5) – an error term modulating a two-factor Hebbian update rule in proportion to the error a given neuron contributes to solving a task, as defined by a loss function

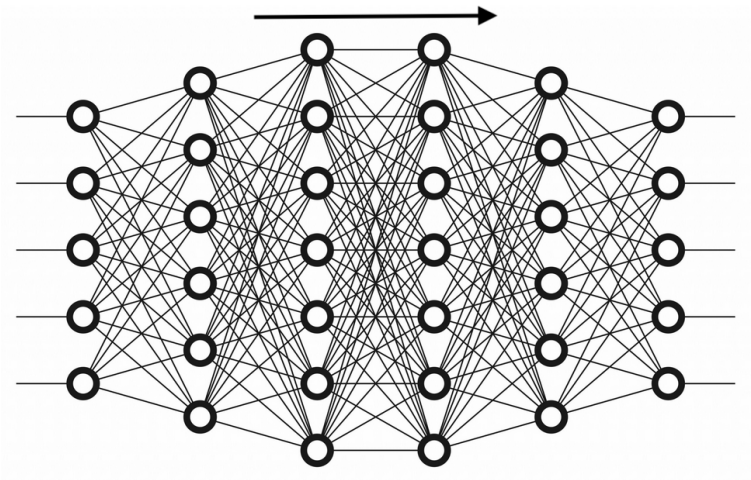


Figure 1.10: Deep feedforward network

[30]. The errors are hypothesized to be computed in the apical dendrites of pyramidal neurons and manifested as dendritic plateau potentials, which have been shown to drive learning *in vivo* [86].

In this way, credit assignment can be thought of as a problem of memory addressing, i.e. selecting which synapses to update in order to produce the desired output in response to a cue (although in this work we focus on the much more narrow scenario where the cue is intended for the network to return a previously stored item rather than solve a pattern recognition task or perform a desired action). When training a network with such an error-driven plasticity rule, however, the synaptic weight updates corresponding to a single item are distributed, incremental, and dependent on the other items being learned. As a result, it is harder to isolate the representation of an item as it appears stored in the synapses of a network, even with full access to the synapses as in the ideal observer model.

Unintentional memorization in deep neural networks

Neural networks are typically not trained explicitly to store and retrieve a set of items the way a SDM or a Hopfield network does (although see Section 1.4.6). Rather, their goal is to approximate a function which convert inputs into a different representation in a generalizable way. For instance, a classifier converts an image into a one-hot vector corresponding to the class, a sequence-to-sequence network converts speech to text, or a generative model converts a query to a

meaningful response. Ideally, the details of the individual training items are discarded in favor of a consolidated representation.

Nevertheless, a neural network with a sufficiently large number of parameters can "memorize" the training set (often, unlike the classical overfitting intuition from statistics, without sacrificing generalization performance [95]). This phenomenon, however does not always correspond to the type of memory we describe here. In the supervised learning scenario, memorization refers to "label memorization" – having zero error on the training set [96, 97, 95], analogous to creating a lookup table. An individual item can also be considered memorized if excluding it from the training data would have a strong impact on the network's ability to classify it [98], i.e. the network cannot generalize to this item from others in the dataset. In this case, if an item is memorized, it is not necessarily the case that it is possible to extract it from the network (although see [99] for an example of adversarial attacks which attempt to do this).

On the other hand, generative models such as variational autoencoders [100], generative adversarial networks [101], diffusion models [102], or large language models [103] that are trained to produce samples from a distribution (which is implicitly defined by the training dataset) have been shown to memorize and return training examples verbatim rather than generating novel ones based on learned statistics [104, 105, 106, 107, 108]. A number of benchmarks have been proposed to detect this type of memorization based on sample outputs [109, 110] or directly querying the network [111], as this type of memorization is undesirable for privacy concerns, and methods have been proposed to prevent it [112].

Memorization by gradient descent

Associative Memories Other work intentionally uses gradient descent to explicitly store and retrieve memories. As originally proposed, Dense Associative Memories or modern Hopfield networks are trained with gradient descent for image classification [68] or multiple-instance learning [113]. We also use this approach in [114] (see Chapter 4).

In [68], the rows of the weight matrix are not a predefined set of memories to be stored. Rather

the "memories" are learned by gradient descent. Although any individual weight update Δw is not interpretable as a memory as it is with ideal observer models, SDM, or Hopfield networks trained with a Hebbian plasticity rule, after convergence of the gradient descent algorithm, each row can be considered a memory and retrieved through content-based addressing as described in Section 1.4.3. To train this network, we present a perturbed input and allow the network to converge to a fixed point which indicates the network's output. We then compute the loss (e.g. mean-squared error or binary cross-entropy) between the output and the unperturbed input and update the parameters according to the gradient of the loss.

Since the gradient depends on the activations of the hidden units, we can consider the incremental weight update to be addressed by the hidden layer in the same way that an update of the contents matrix for the SDM is addressed by its hidden layer. Indeed, in Chapter 5, we consider computing the gradient in a MHN with recurrent backpropagation [37, 38]. Taking the first-order term of this expression, we can show that the gradient update (approximately) corresponds to a Hebbian plasticity rule, with the pre-synaptic term being the error in the output units and the post-synaptic term being the hidden layer activity (along with higher-order correction terms). In this network, however, the hidden activity is determined by content-based addressing, so this training algorithm can be thought of as content-addressed writing or storage (in comparison to content-addressed recall described in Section 1.4.3).

Nevertheless, since the training set size is larger than the pure memorization capacity of the network, the inputs are correlated, and training is done with an iterative gradient-based scheme, the network learns consolidated representations of the data (either features or prototypes [68], depending on the sharpness of the hidden layer nonlinearity; see Section 1.4.4 and Chapter 4) rather than verbatim memorization. In Chapter 4. we extend this idea to train a MHN to perform concurrent memorization and consolidation. If an item occurs frequently, it should be consolidated and stored in a distributed manner, so that common features are extracted and fine-grained details discarded. If it is rare, it should be memorized verbatim since there is not enough information about which details are dispensable and which constitute the item's essential defining features. To do so, we propose

a tunable content-addressed writing scheme, controlled by setting the learning rate low or high, respectively, corresponding to incremental distributed learning or rapid few-shot memorization of a localized representation [114].

In [113], the authors use a different approach, where the stored memories are indeed (learned embeddings of) individual inputs. "Memory Networks" [115, 116] discussed in Section 1.4.7, as well as examples from reinforcement learning (RL) [117, 118, 119] use a similar memory structure that sequentially stores the entire input history for an episode. Although the authors do not specify an explicit network mechanism for writing, a three-factor rule as in [21] with sequential third factors can be used (see Section 1.4.5). In this case, rather than the memory contents, the *query* is learned by gradient descent (this is different from Memory Networks or memory-augmented RL networks, where the query is either dynamically generated from the input or by the "controller" network), but the lookup mechanism is still an explicit content-based address as in Section 1.4.3.

Autoencoders Autoencoders with a single hidden layer, which have a network architecture similar to the KVM (with keys equal to the values, by definition of the *auto*-encoder), can be interpreted in the same way: rows of the encoder matrix are content-based memory addresses (keys), and the columns are the corresponding values, although both are now learned through gradient descent rather than stored through a single update. In this way, the hidden layer representation is analogous to the coordinates for a basis that consists of the columns of the hidden-to-output layer – an address in the space of memories, allowing linear combinations of stored items to be recalled. The latent space of deep autoencoders with multi-layer encoding and decoding functions can be considered an "address" in the same way, although the individual memories are harder to define because they can be hierarchical or fully distributed as discussed above. To complicate matters further, because their decoders are nonlinear, combinations of latent space neurons do not correspond to linear combinations of their individual outputs.

Autoencoders can be trained in several configurations. A denoising autoencoder [120] is trained similarly to the Dense Associative Memory described above – a perturbed item is given as the

input, the loss between the its recovered ("denoised") output and the original unperturbed target is computed, and a gradient update step is performed. A sparse (e.g. [121]) or undercomplete [122] autoencoder uses the unperturbed item as both the input and target, but introduces a bottleneck in the hidden layer – either sparsity or a dimensionality reduction, respectively – to prevent the network from learning the trivial feed-through (identity) function. Variational autoencoders [100] introduce noise in the hidden layer. This encourages continuity in the latent space and, by randomly sampling from it, enables the network to generate novel outputs from the same distribution as the inputs. In this way, memorization of items and generation of new ones of the same type can be seen as part of a continuum – indeed, similarities between generative diffusion models and associative memories have recently been discussed [123, 124].

As a result of this training, although in the dataset the input keys are equal to the output values, their representation in memory (rows and columns of the encoder and decoder matrices) in general is not. This is in contrast to the MHN where the rows of the feedforward/encoder/addressing matrix are, by definition, the transposes of the columns of the feedback/decoder/contents matrix. From a perspective of memory addressing, this can be understood as the encoder being optimized for content-based memory lookup, which is not necessarily the optimal representation for memory retrieval. Interestingly, overparameterized autoencoders, where the dataset size is much smaller than number of network parameters, have been shown to behave similarly to Hopfield networks despite their lack of explicitly enforced weight symmetry [125]. Recursively feeding the autoencoder output back into the input leads to fixed-point dynamics, where the fixed point is exactly one of the memories that was learned by the autoencoder.

1.4.7 Memory addressing in artificial intelligence

Thus far, we have discussed neural networks designed to be models of biological memory, and the memory addressing mechanisms that they propose. In recent years, neural networks used for artificial intelligence have been augmented with explicit memory structures that can be used to read and write intermediate information during computations.

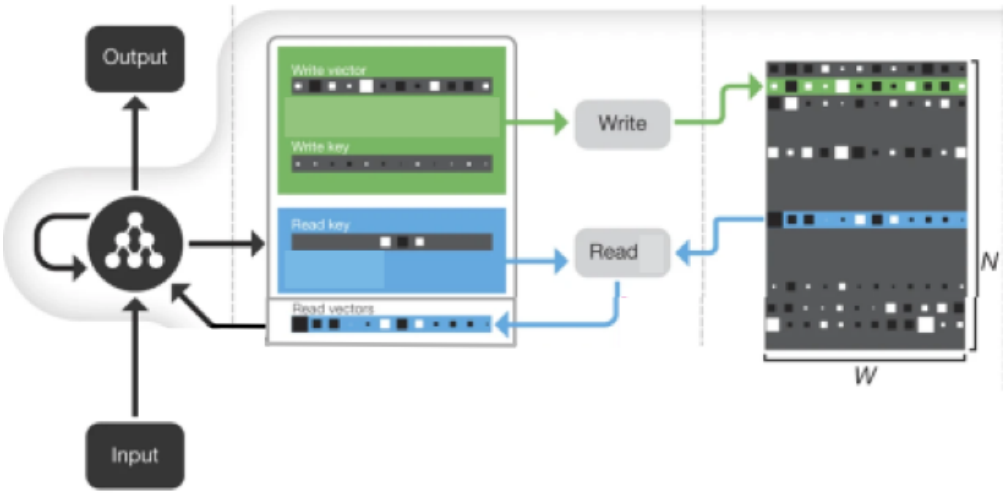


Figure 1.11: Simplified Differential Neural Computer / Neural Turning Machine architecture. A recurrent neural network "controller" (left) interacts with an external memory bank (right) by generating read and write keys which are transformed into a content-based address for indexing into the memory. Adapted from [126].

Neural Turing Machines [127], later improved and referred to as the Differentiable Neural Computer [128], consist of an RNN "controller" that interacts with an external memory through read and write operations (Figure 1.11). The memory is instantiated simply as an $N \times M$ matrix, and can be interpreted as synaptic weights analogous to those of the contents matrix C of the SDM [59] (see Section 1.4.2) with n memory slots, each containing a vector of length m (although see [129] for more complex instantiations of the memory bank). In addition to the standard recurrent dynamics of updating the hidden unit state, at each timestep the RNN receives a length-m "read" vector from the memory bank which is used as an input to the RNN at the next timestep, and generates length-m "add" and "erase" vectors which define the contents to be written or removed from the memory bank.

Critically, the RNN also generates a length-n addressing vector, analogous to the hidden layer of the SDM. At each timestep, a length-m "key" vector is generated and used as a content-based address into the memory matrix (see Section 1.4.3). This returns a length-n vector subsequently sharpened by the softmax function, the specificity of which is controlled by an inverse-temperature parameter (see Section 1.4.4) also generated at each timestep by the controller. As a second step, to add a location-based addressing component, the addressing mechanism allows the possibility to circularly shift the content addressing vector, enabling the weighting to iterate over a sequence of

addresses in consecutive timesteps. The result is a length-n addressing vector that is used to index the locations of the memory matrix for both reading and writing.

Memory networks [115, 116] use a similar approach applied to textual reasoning tasks. In this model, the authors store sentences (embedded in a continuous vector space) in memory slots sequentially, avoiding the need for complex addressing mechanisms for writing. Given an embedding of a sentence query, the model uses the stored information to find the appropriate response. This addressing mechanism for retrieval is also simpler, using purely content-based addressing by comparing the query to stored keys via the dot product, and either a hard threshold address selection mechanism [116] analogous to WTA or k-WTA (see Section 1.4.4), or its differentiable analogue, the softmax [115]. Similar networks have also been used to enhance the memory capabilities of reinforcement learning agents [117, 118, 119].

Memory-Augmented Neural Networks [87] use the same content-addressable reading mechanism with a softmax but, unlike the NTM, use distinct addressing vectors for reading and writing. The distinct write-addressing vector allows the controller to interpolate between writing to the most recently accessed memory slot (given by the read vector), or replacing the contents of the least recently written-to slot. To solve the problem of computational overhead due to large memory sizes, [130] use a similar network, but enforcing sparsity in the read vector (which implies sparsity in the write) through a k-WTA function. Note, however, that this is unlikely to be a problem in biological networks which would query or update all of the memory slots in parallel without having to rely on the sequential processing in a CPU.

In this literature, the authors do not specify a biological mechanism for reading from and writing to the memory bank. In [21] (Chapter 3), we propose an implementation of this type of memory that has an explicit network architecture for readout of information, along with biologically plausible three-factor learning rules for storage.

Other work has also considered using ongoing plasticity dynamics, referred to as "fast weights" [131] in an RNN as an auxiliary memory structure. In addition to the usual recurrent dynamics, the authors propose that the synaptic weights are updated at each timestep according to a Hebbian rule.

In this case, however, this memory structure is not a flexible input/output memory the way that the NTM memory is, and does not have an explicit mechanism for addressing reads and writes. Rather, it is "equivalent to attending to past hidden vectors in proportion to their scalar product with the current hidden vector" [131]. In [132], the authors also meta-learn the relative weightings of the Hebbian component of each synapse, which can be interpreted as a static addressing mechanism, but the exact operating principles of which are opaque.

Finally, note that even "vanilla" RNNs [133] without any explicit memory mechanism can store memories through persistent recurrent activity[3]. Gated RNNs such as LSTMs [91] or GRUs [135] make this explicit through additional structure which allows them to clamp values in a gated memory cell over time. However, there is again no transparent addressing mechanism to access the stored input, other than the dynamics of the RNN which can change the internal representation of a stored item over successive timesteps. In fact, there is no reason *a priori* that memory-augmented RNN controllers or plastic RNNs cannot use persistent activity as a memory mechanism in addition to (or instead of) the dedicated memory bank. Conversely, is also possible that a gated RNN uses the gates simply as an auxiliary computational structure rather than a memory (although see [136]), making the question of memory addressing difficult to analyze.

1.4.8 Memory addressing in biological systems

Compared to computational models, memory addressing mechanisms in biological organisms are more difficult to analyze because they cannot be as readily isolated. As discussed in section 1.1, unlike computer systems, neural networks generally perform computation and memory storage and retrieval using shared hardware. Rather than a dedicated processor operating on inputs retrieved from memory, the memory may define or modulate the computation itself. Even if there exist dedicated "memory" and "compute" subsystems, they are more difficult to disentangle as they may be co-located and not readily differentiated based on taxonomies such as cell types or anatomical

[3]Note that in this case, "memories" refer to "stored input history", limited by the network's "input capacity", controlled by the number of dynamic variables in the model. This is different from a network's "task capacity" which limits is the information extracted from a training dataset, and is controlled by the number of tunable parameters in the model [134].

features. Moreover, unlike neural recordings, we do not have large-scale experimental access to the dynamics of synaptic weights *in vivo*. This makes it difficult to directly fit plasticity rules [137, 46], or even establish which synapses are plastic. Nevertheless, there is some evidence to support that biological neural networks may follow some of the memory addressing principles described in this work.

Engram cells and cell assemblies

Once deemed elusive [138], engrams are now considered to be the biological substrate for memory in the brain. Understanding their formation and subsequent retrieval can provide insights for refining the computational models of memory addressing discussed so far, building more powerful memory structures for artificial intelligence, and enabling translational research into human memory in health and disease. Generically, "engrams" refer to any of the long-lasting physical changes in the brain as a result of learning, whether these changes are synaptic or neuronal, implemented on a molecular, protein, or cellular level [139]. In practice, the engram literature is centered around "engram cells" which comprise interconnected "engram cell assemblies". These are populations of cells that are activated during a learning event, and which are subsequently reactivated during its retrieval.

Gain- and loss-of-functions studies have demonstrated the sufficiency and necessity of engrams for memory-driven behavior [140, 141]. In a natural setting, an engram cell assembly would be reactivated by an external cue, such as a particular context for a fear conditioning experiment where a context (an experimentally chosen combination of textures, colors, shapes in the surrounding environment) gets paired with a foot shock. Being placed in the same context again causes expression of a fear memory – in a mouse, this corresponds to freezing behavior.

It is possible, however, to induce freezing in a neutral context (not paired with a foot shock) by optogenetically *reactivating* the cells that were active during the fear conditioning procedure, which correspond to the cell assembly representing that memory [142]. Conversely, it has been shown that *silencing* the engram cells when placed in the foot-shock context reduces the amount

of freezing, therefore (putatively) suppressing the fear memory [143]. It is also possible to create "silent engrams" that cannot get reactivated by natural cues but can be reactivated by artificial stimulation of the neural ensemble. These can be created through blocking of protein synthesis after the learning event, behavioral extinction, or simply memory decay.

In this way, engram retrieval is reminiscent of content-addressable memory. Partial activation of an ensemble reinstates the full memory, like presenting a perturbed input restores the original vector through fixed-point dynamics of a Hopfield network. Suppressing the ensemble activity for a specific memory suppresses that memory's expression, but not others, as long as the ensembles are disjoint. Finally, silent engrams, due to their inability to be activated by natural cues, demonstrate a problem of memory addressing. These neurons cannot be activated by the brain's intrinsic addressing mechanisms (e.g. through the feedforward weights of a MHN), but knowing the "address" (set of hidden neurons) in advance and activating those specific neurons shows that the memory is still encoded[4] (e.g. by reinstating it it through the feedback weights of an MHN). It is, however, unclear whether the network architecture is that of a classical Hopfield network with implicit addressing, a modern Hopfield network where explicit addresses can be found, or another associative memory network structure.

The process of engram formation is the process of memory writing. Although plasticity rules have not been investigated specifically in the context of engrams, it is plausible that they are created through some form of Hebbian plasticity. First, it has been shown that neurons which are more excitable are more likely to be "allocated" to an engram during an encoding event [144]. Those that are allocated also show an increase in dendritic spine density. This is consistent with Hebbian plasticity, as neurons that are more likely to fire are also more likely to strengthen synaptic connections among them. This holds true whether their elevated excitability is experimentally controlled or intrinsic [141]. In the latter case, it can be due to random variation, or to selection of those neurons through one of the addressing mechanisms discussed in this work. A similar motif is

[4]An alternative explanation is that there is no silent engram and the experimenter is simply activating a representation of a fearful context in an absolute sense rather than reactivating a memory. In other words, as a thought experiment, if we were to optogenetically activate that same "silent engram" set of neurons *without the mouse ever having experienced the fear conditioning paradigm*, the alternative hypothesis is that the mouse may still exhibit freezing behavior.

discussed in Section 1.4.4 – inhibitory neurons control a neuron's excitability to decorrelate it from other neurons' activities [74].

It has also been shown in a biophysical model that in addition to intrinsic excitability, engram formation is mediated by inhibitory competition [145]. Regardless of the overall level of excitability, the same number of neurons are recruited for the engram. To explain this observation, the authors suggest that the most excitable neurons suppress the rest of the candidate cells through inhibitory interneurons. Indeed, this may be an instantiation of the content selection mechanism described in Section 1.4.4, in the form of either the softmax or the k-WTA function.

Hippocampus as an addressing system

Originally proposed by [146], the hippocampal memory indexing theory suggests that while the neocortex stores information about an event, the hippocampus serves as a "coordinate system", "index", or "map" of the cortical modules that were active during the event. The hippocampus stores only the spatiotemporal sequencing of cortical modules, and not an neural encoding of the event itself. In this framework, memory storage occurs when the neocortex, representing an episode to be remembered, activates a set of hippocampal synapses which are strengthened as a result. During recall, a *partial* activation of the neocortical representation can activate the corresponding "index" in the hippocampus, the synapses of which were potentiated during storage. The hippocampus then projects back to the neocortex to reinstate the full activation, thus retrieving the memory in a content-addressable manner (Figure 1.12).

In this view, the hippocampus can be modeled by the hidden layer (i.e. the addressing function) of one of the recall memory networks discussed here – the SDM (Section 1.4.2), MHN (Section 1.4.3 and Chapters 4 and 5), or KVM (Sections 1.4.3 and 1.4.5 and Chapter 3) – while the cortex is represented by the input/output layers. The two-layer model of the modern Hopfield network, however, is much simpler than the structure in the hippocampo-cortical circuit, which consists of anatomically and functionally distinct sub-regions including the entorhinal cortex, dentate gyrus, CA3 and CA1. Nevertheless, despite differences (or abstractions) in the model implementation, the

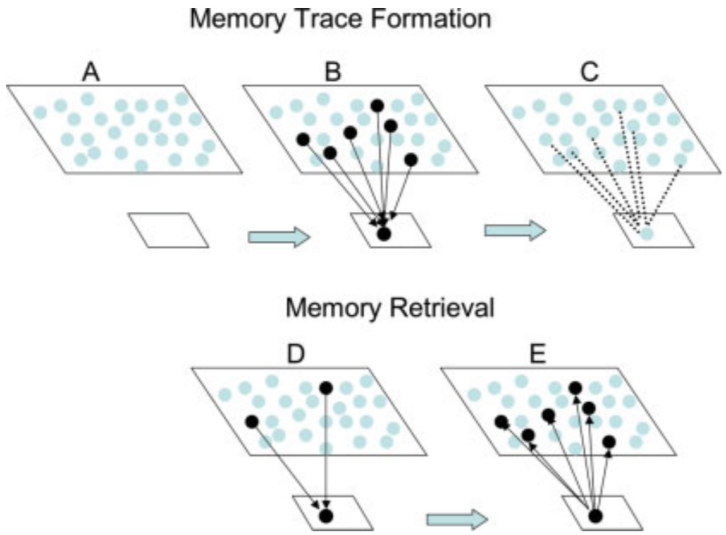


Figure 1.12: Memory storage and retrieval mechanisms in the hippocampal memory indexing theory closely resemble the bipartite structure of the modern Hopfield network, where the visible (input/output) layer corresponds to the cortex (large top square) and the hidden layer corresponds to the hippocampus (small bottom square). Image from [147].

computational principle of indexing remains the same. To incorporate this anatomy, [147] suggest a hierarchical indexing structure, where the hippocampus does not index the neocortex, but rather the entorhinal cortex, which then in turn indexes the neocortex. Following this structure, it is possible to extend the MHN to incorporate multiple layers, known as the hierarchical associative memory [148].

A modern look on the hippocampal indexing theory connects it to the results seen in engram studies. In [149], the authors suggest that hippocampal engrams can be exactly interpreted as a hippocampal index. Notably, in a contextual fear conditioning paradigm, reactivating engram-tagged cells in sensory cortices does not reinstate behavior as efficaciously as hippocampal engrams, suggesting their importance as an index [150]. Other work [151] observes only a subset (~25%) of place cells which function as engram cells (identified as expressing *c-fos*, an immediate early gene activated by neural activity) during memory formation. Similarly, a study in the hippocampus of food-caching birds [152] shows a dissociation between "barcode" cells which uniquely encode individual memory episodes, and cells encoding location or food identity. These results suggest a distinct role for a subset of hippocampal neurons in instantiating an index which reactivates

either a contextual memory [151] or a food-caching event [152]. The authors in [149] furthermore propose that hippocampal dysfunction due to injury or psychiatric illness might be interpreted as an "indexopathy" – an inability to perform "hippocampal-dependent information routing" – in the same way that silent engrams are a problem of retrieval rather than the absence of a memory.

1.5 Conclusion

In both biological and artificial neural networks, memory storage can be broken down into two complementary but interrelated components. The first is synaptic plasticity – modifications to synaptic weights in response to an input stimulus, calculated by a rule that depends on the network's neural activity, input history, and/or a target signal. The second is memory addressing – the function by which the synapses that should be updated are selected during memory storage, as well as a (potentially different) function by which those synaptic updates are retrieved and decoded back into the neural representation of the original input stimulus or, in the case of familiarity, a signal indicating that the stimulus has been previously encountered.

In this chapter, we have discussed a number of mechanisms by which this selection can be performed. First, fixed strong feedforward weights can operate on an input "query" as an "address decoder" to activate a set of neurons that serve as the memory's "address". Synapses efferent from these neurons can be updated to store a desired stimulus through simple Hebbian plasticity controlled by pre- and post-synaptic neural activity. Retrieving memories stored in this manner is similar – given the same query, the corresponding address is selected and the stored memory is retrieved through feedforward activation (Chapter 2).

Second, rather than a fixed address decoder function, the address for memory retrieval can be selected by comparing the query with a modifiable set of "keys". Known as content-addressable memory, this function selects the address which maximizes the overlap (usually measured as the dot product or cosine similarity) between the keys and the query.

In this case, memory storage can proceed in one of two ways. The simplest is through a local third factor which, independently of the memory contents, selects a slot to be written or updated

(Chapter 3). Alternatively, storage can be done in a content-addressable manner as well. A forward pass selects the address (hidden layer activation), and the corresponding synapses are updated with a gradient descent step ([114] and Chapter 4). In the modern Hopfield network, such gradient descent methods can also be approximated by a Hebbian plasticity rule, where the pre-synaptic factor is not neural activity but an error (Chapter 5).

In each case, a mechanism for refining the specificity of the address is necessary. Implemented as a neuronal nonlinearity, this can be either a simple threshold – if an overlap between the query and a key in the address decoder is sufficiently large then the corresponding address bit (neuron) is activated – or a more complicated function where all the inputs to the address vector (hidden layer) are compared and either the maximum is taken (winner-take-all), or they are sharpened in an adjustable manner, as a "soft" version of the maximum (softmax).

By explicitly highlighting the function and mechanisms of memory addressing, we hope to separate this component of memory storage from that of synaptic plasticity. Although not arbitrarily, addressing mechanisms for storage and retrieval can be combined with each other and with different plasticity rules. For example, we can consider using complex synapses [121] together with a sparse distributed memory, or a key-value memory network for an improved storage capacity. Gradient-based synaptic updates can be combined with a meta-learned sparsity mask which selects a subset of synapses to update through gradient descent [153]; in the view of memory addressing, the sparsity mask can be interpreted as an address to control the location of synaptic plasticity.

Finally, we discussed applications of these addressing mechanisms for dedicated memory devices in artificial intelligence systems, and considered experimental evidence to support their existence and functionality in biological organisms. A deeper understanding of memory addressing can help us improve security in artificial intelligence systems by selectively removing memories of private data, as well as aid in treatment of "indexopathies" [149] in biological organisms.

Chapter 2: Meta-learning synaptic plasticity and memory addressing for continual familiarity detection

The work presented in this chapter was done in collaboration with Guangyu Robert Yang and LF Abbott. The text is an unabridged version of the work published in [2]. We thank Stefano Fusi, Ken Miller, Dmitriy Aronov, James Murray, Marcus Benna, SueYeon Chung, Juri Minxha, Taiga Abe, and Denis Turcu for helpful discussions.

Over the course of a lifetime, we process a continual stream of information. Extracted from this stream, memories must be not only efficiently encoded but also stored in an addressable manner for subsequent retrieval. To explore potential encoding and addressing mechanisms, we consider a continual familiarity detection task in which a subject must report whether an image has been previously encountered. We design a class of feedforward neural network models endowed with biologically plausible synaptic plasticity dynamics and a static addressing matrix, both of which are meta-learned to optimize familiarity detection over long delay intervals. After training, we find that anti-Hebbian plasticity leads to better performance than Hebbian and replicates experimental results from the inferotemporal cortex, including repetition suppression. A combinatorial addressing function also emerges, selecting a unique neuron as an index into the synaptic memory matrix for storage or retrieval of a given stimulus. Unlike previous models, this network both operates continuously without requiring any synaptic resets and generalizes to intervals it has not been trained on. We demonstrate this not only for uncorrelated random stimuli but also for images of real-world objects. Our work suggests a biologically plausible mechanism for continual learning, and demonstrates an effective application of machine learning for neuroscience discovery.

2.1 Introduction

Every day, a continual stream of sensory information and internal cognitive processing causes lasting synaptic changes in our brains that alter our responses to future stimuli. It remains a mystery how neural activity and local synaptic updates coordinate to support distributed storage and readout of information and, in particular, how ongoing synaptic changes due to either new memories or homeostatic mechanisms do not interfere with previously stored information.

Memory research in theoretical neuroscience and machine learning has addressed these questions through modeling studies, but important features remain to be clarified. First, memories of an individual's history are encoded in a one-shot manner – this is different from typical neural network models which use a prolonged incremental training process to learn a complex task. Such training algorithms use a global error signal and perform per-synapse credit assignment through knowledge of the entire network [27], whereas biological synapses typically only have access to local pre- and postsynaptic activity [17] and various modulatory signals [83, 20]. Second, biological synapses change continually in response to ongoing activity, whereas models commonly assume that synapses are fixed after training ends, such as the classical Hopfield network [64] and most deep neural networks [154]. Unregulated continual updating of synapses can cause catastrophic forgetting in which a network either erases previous memories [44, 45] or renders stored information unreadable [155]. Recurrent neural networks are commonly used to perform tasks that involve memories sustained by neural activity [133, 91, 156], however most memories are likely stored through synaptic potentiation and depression [16]. Synaptic memory has sometimes been studied through an ideal observer approach [43, 49] in which synaptic weights are directly accessible for readout, but biological organisms must read out synaptic storage through neuronal activations. In fact, the readout may not be a dedicated circuit as in attractor network models of memory [64], but rather manifested as a change in ongoing neural processing [157]. Finally, machine learning research often eschews biological plausibility and mechanistic understanding in favor of performance on benchmark tasks [131, 158, 127, 159, 132, 85].

Familiarity detection – identifying whether a stimulus has been previously encountered – is a simple and ubiquitous form of memory that serves as a useful testbed for addressing these issues. Classical studies have demonstrated that human recognition memory capacity for images is "almost limitless" in a two-alternative-forced-choice task with separate encoding and testing phases: retention follows a power law as a function of the number of items viewed [160]. Pioneering theoretical work has shown that the number of memories stored by a familiarity detection network depends on the synaptic plasticity rule and, in the case of uncorrelated inputs, capacity can scale proportionally to the number of synapses [161]. More recent behavioral work further demonstrates an impressive capacity in a continual setting, the error rate as a function of the number of intervening items exhibiting a "power law of forgetting" [162]. Theoretical studies have shown that power-law forgetting is achievable by synapses with metaplasticity, using both uncorrelated inputs [49] and face images [163]. Neural signals of visual familiarity have been observed as reductions in responses to repeated presentations of a stimulus, a phenomenon known as repetition suppression [164, 165, 166, 167]. At the timescales relevant for this task – one-shot memorization on the order of seconds and long-term forgetting on the order of days – this is plausibly caused by depression of excitatory synapses or potentiation of inhibitory ones [46].

Previous modeling work on recognition memory used a predesigned architecture and plasticity rule and both empirical and analytic evaluation of performance [168, 161, 169, 170]. An emerging approach employs a machine learning technique known as "meta-learning," or "learning how to learn" [171], that uses optimization tools to rapidly search for mechanisms that artificial neural networks can use to solve a learning/memory task. In contrast to hand-designed models, meta-learning enables exploration of a large family of architectures and plasticity rules and reduces the bias inherent in human guesswork. Importantly, it is possible to impose constraints that ensure biological plausibility [172]. For example, given a spiking or rate-based network architecture and a family of biologically plausible plasticity rules with tunable parameters, the meta-learning algorithm can search for the optimal plasticity rules that enable the network to solve a supervised [173, 174], semi-supervised [175], unsupervised [176, 173, 177], or reinforcement learning [173, 178] task or – as in our case –

to store the input history for subsequent use. The meta-learning algorithm itself can be a variation of gradient descent [175, 174], an evolutionary strategy [176, 173, 178], or another optimizer depending on the nature of the network and the task.

In this work, we investigate not only "how" memories are stored – the synaptic plasticity rule – but also "where" – the mechanism for addressing the storage and retrieval locations. Classical models of memory rely on "content-based addressing" [64], where a partial cue elicits recall of the full memory through recurrent dynamics, but do not explicitly select which synapses store the memory. On the other hand, "key-value" memory networks in machine learning [127, 159], store values in a memory matrix indexed explicitly by keys, analogous to the addressing in a computer random-access memory (RAM), although such models lack a biological interpretation [21]. Our model includes both a synaptic plasticity rule and a realistic addressing mechanism.

Positing that the answer to "when" plasticity should occur is "always", we consider a simple version of "what" to remember – familiarity. We construct a family of models that recognize previously experienced stimuli and, importantly, learn and operate continuously without separate learning and testing phases. The capacity of these networks remains constant over time, so they can be continually fed new inputs with no reduction in steady-state memory performance.

We use a feedforward network architecture with ongoing Hebbian plasticity in its synaptic weights, parameters of which are meta-learned using gradient descent to optimize the continual familiarity detection process. To isolate synaptic plasticity as the unique memory mechanism, we avoid recurrent connectivity that could store memory through maintained neuronal activations. This architecture, unlike recurrent networks, generalizes naturally over a range of repeat intervals even if trained on a single interval. We show that an anti-Hebbian plasticity rule (co-activated neurons cause synaptic depression) enables repeat detection over longer intervals than a Hebbian rule, and this is the solution most frequently found by meta-learning. This rule leads to experimentally observed features such as repetition suppression in the hidden layer neurons. Furthermore, an addressing function emerges through strong static feedforward weights, selecting a unique neuron to index the synapses for storage of a novel stimulus and detection of a familiar one.

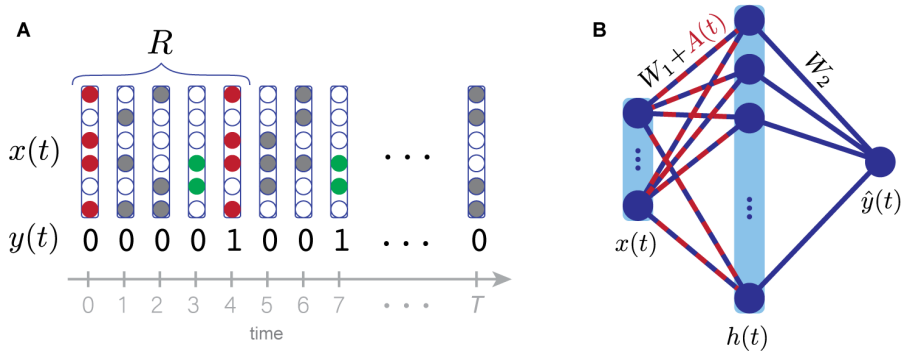


Figure 2.1: Continual familiarity detection task and HebbFF model. (A) The continual familiarity detection task. Given a continual stream of stimuli $x(t)$, the desired output is $y(t) = 1$ if the stimulus has appeared previously and $y(t) = 0$ otherwise. For a given dataset, repeat stimuli always appear at an interval R after their first presentation. Although the task is continual, for the purposes of network training we use a finite-duration trial of length $T \gg R$. (B) The HebbFF network architecture. A feedforward layer is endowed with ongoing Hebbian plasticity, the parameters of which are optimized using stochastic gradient descent. The hidden units are linearly read out to produce the network's estimate of familiarity $\widehat{y}(t)$.

2.2 Results

2.2.1 Continual familiarity detection task

We consider a continual familiarity detection task (Figure 2.1A) in which a stream of stimuli is presented to a network. With probability $1 - p$, the stimulus at time *t* is chosen as a randomly generated d-dimensional binary vector $\mathbf{x}(t)$, where each component is either $+1$ or -1 (note that for sufficiently large d, spurious chance repeats are extremely unlikely). With probability p, the stimulus is a copy of the stimulus presented R time steps ago, so that $\mathbf{x}(t) = \mathbf{x}(t - R)$. However, we ensure that a stimulus is repeated at most once so, if $\mathbf{x}(t-R)$ is already a repeat, i.e. $\mathbf{x}(t-R) = \mathbf{x}(t-2R)$, a new $\mathbf{x}(t)$ is generated. As a result, the fraction of novel stimuli, which we call f, is not equal to $1 - p$, but rather $f = \frac{1}{1+p}$. We use $p = \frac{1}{2}$, so $f = \frac{2}{3}$. The output of the network should be $y(t) = 0$ if $\mathbf{x}(t)$ is novel and $y(t) = 1$ if it is familiar, i.e. has appeared previously.

The accuracy of the network ($P_{correct}$, the probability of correctly responding to a stimulus) depends on two factors: the true positive rate (P_{TP}, the probability of correctly reporting a repeated stimulus as "familiar"), and the false positive rate (P_{FP}, the probability of incorrectly reporting a

novel stimulus as "familiar"). These two factors are weighted by the fraction of novel stimuli f, so that $P_{correct} = (1-f)P_{TP} + f(1-P_{FP})$. Through our choice of loss function (section 2.4), we are effectively training the networks to maximize accuracy, so the "chance" level performance is f (for $f > \frac{1}{2}$), which a network can achieve by reporting all stimuli as novel ($P_{TP} = P_{FP} = 0$).

In our paradigm, a given dataset has a single repeat interval R, which differs slightly from previously studied experimental paradigms [162, 165]. However, we evaluate performance on multiple datasets with various values of R. For testing, this is analogous to evaluating a single dataset with multiple repeat intervals and computing accuracy for each interval separately. We use this approach because it allows us to test generalization by training on one value of R and testing on others. It also allows us to train the network to its maximal capacity by gradually increasing R during "curriculum training", and simplifies analytic calculations.

We begin by considering familiarity detection for uncorrelated stimuli, but, in later sections, we generalize to a task that requires simultaneous familiarity detection and binary classification, and to a dataset of real-world object images.

2.2.2 HebbFF network architecture

To investigate the effectiveness of synaptic plasticity for solving this task, we use a feedforward neural network with a single hidden layer and, to implement the memory function, activity-dependent ongoing Hebbian plasticity (HebbFF) (Figure 2.1B). We purposefully do not include any recurrent connections to ensure that memory cannot be stored through persistent neuronal activity, thus isolating synaptic plasticity as the only possible memory mechanism.

In the HebbFF network, a group of hidden layer neurons with firing rates given by an N-dimensional vector $\mathbf{h}(t)$, receives a d-dimensional input $\mathbf{x}(t)$. The variable t indexes stimulus presentations which occur sequentially, so we refer to it as "time". The input to each hidden-layer neuron is weighted by its corresponding synaptic strength and then transformed into a hidden-layer firing rate through a nonlinear activation function $\sigma(\cdot)$. The synaptic strength between the postsynaptic neuron with rate $h_i(t)$ and the presynaptic neuron carrying the input $x_j(t)$ is the (i,j)

component of an N-by-d matrix that is the sum of a fixed matrix $\mathbf{W}_1$ and a plastic matrix $\mathbf{A}(t)$. Thus, the firing rate of the hidden layer is given by

$$\begin{aligned}\mathbf{h}(t) &= \sigma\left((\mathbf{W_1} + \mathbf{A}(t))\,\mathbf{x}(t) + \mathbf{b_1}\right)\\ \mathbf{A}(t+1) &= \lambda\mathbf{A}(t) + \eta\mathbf{h}(t)\mathbf{x}(t)^T\end{aligned}$$

Finally, the output of the network $\widehat{y}(t)$ is a linear readout of the hidden layer and, since the target $y(t)$ is binary, we bound the readout with the logistic function,

$$\widehat{y}(t) = \sigma(\mathbf{W_2}\mathbf{h}(t) + \mathbf{b_2})$$

The response of the network is "familiar" if $\widehat{y}(t) > 1/2$ and "novel" otherwise. Although in the general case $\mathbf{W_2}$ is unconstrained, to simplify analysis we later consider a uniform readout where all entries of $\mathbf{W_2}$ are equal, with no appreciable change in performance.

To construct the network, we use backpropagation through time (BPTT) to "meta-learn" the parameters $\mathbf{W_1},\ \mathbf{b_1},\ \mathbf{W_2},\ \mathbf{b_2},\ \lambda,\ \eta$, which are fixed once training is completed (section 2.4). The continual familiarity detection task – the "learning" task – is then performed exclusively by the ongoing synaptic dynamics of $\mathbf{A}(t)$, determined by the fixed parameters. These dynamics are a biologically plausible mechanism for solving the continual memory task, but BPTT is simply used as an optimization tool to find suitable parameters of the network. Although the optimization mechanism is non-biological, we can nevertheless interpret the learned fixed parameters as changing over much longer timescales (generations or years) than those of the plastic matrix (minutes or seconds).

2.2.3 The "what" of synaptic plasticity: encoding via an outer product for generalization

As a benchmark for comparing HebbFF performance, we first train a long short-term memory (LSTM) network [91] – a recurrent neural network (RNN) architecture well-suited for memory performance – on the continual familiarity detection task. Unlike HebbFF, which stores its input

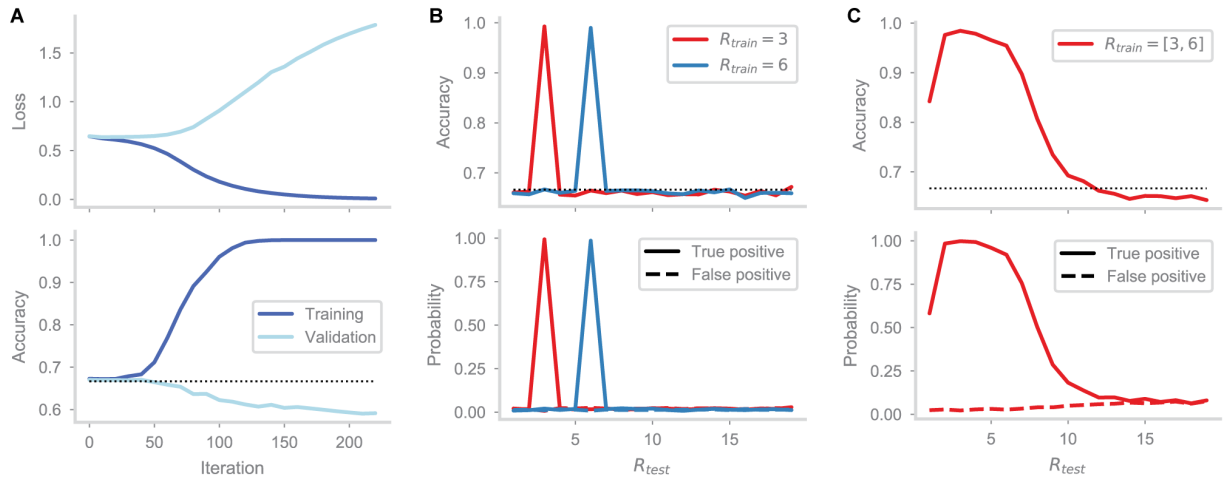


Figure 2.2: RNN performance on continual familiarity detection. (A) Training an LSTM ($d = 100$ input dimension, $N = 100$ recurrent units) on a single dataset of a familiarity detection task ($T = 500$ stimulus presentations, repeat interval $R = 3$). Although the loss (top) approaches zero and accuracy (bottom) approaches 1 on the training dataset (red curves), performance on a validation dataset (blue) with the same parameters fails to generalize even when tested in-distribution with the same R. (B) Training the RNN using "infinite data." New datasets, each with $R = 3$, are generated at every epoch of training (red). Accuracy (top), as well as true positive and false positive probabilities (bottom) is shown as a function of the repeat interval on validation datasets. The same is repeated with another RNN using $R = 6$ (blue). The RNNs perform well in-distribution on datasets with the same repeat interval as used during training, but fail to generalize out-of-distribution to other repeat intervals. (C) Training the RNN with "infinite data," using datasets with both repeat intervals $R = 3$ and $R = 6$. The RNN interpolates between the intervals – performance is high when tested on repeat intervals $3 \leq RR \leq 6$ – but fails to extrapolate. Performance quickly drops for longer repeat intervals, and even for shorter ones.

history in the plastic synaptic matrix $\mathbf{A}(t)$, an RNN uses ongoing neuronal activity.

If we train the RNN using a single dataset with $T = 500$ image presentations (section 2.4) and a repeat interval of $R = 3$, it successfully learns the training set, but entirely fails to generalize to new test sets with the same R (Figure 2.2A). To fix this, we use an "infinite data" approach in which we generate a new dataset for every iteration of BPTT, each with the same value of $R = 3$. Trained in this way, the RNN now generalizes "in-distribution" across datasets with $R = 3$ (i.e. to datasets drawn from the same distribution as the training data, which is parameterized by R), but fails to generalize "out-of-distribution" to data with any other value of R (i.e. to datasets from a different distribution) (Figure 2.2B). The same result holds with $R = 6$ (Figure 2.2B). We can further train the RNN with items spaced at intervals of both $R = 3$ and $R = 6$ (i.e. the value of R is chosen randomly for each familiar stimulus rather than being fixed). While the network can interpolate between the trained values, it does not extrapolate well to larger or smaller ones (Figure 2.2C). Although it is likely possible to train the RNN to perform well for multiple values of R by using more complex training schedules, we believe that poor out-of-distribution generalization is a bottleneck of the RNN approach.

In contrast, the HebbFF network exhibits both in-distribution and out-of-distribution generalization. Even when trained on a single dataset with a fixed repeat interval R, the network generalizes not only to new test sets with the same R, (Figure 2.3A) but even to those with different R values. Critically, the training procedure is the same as for the RNN above, but HebbFF successfully learns a qualitatively different solution due to its better inductive bias. Trained with "infinite data" (the scheme we use in general), HebbFF generalizes to datasets with smaller and even larger R values (Figure 2.3B). If we match the number of dynamic variables rather than the number of hidden neurons, HebbFF still shows superior generalization compared to the RNN (Figure 2.8). This qualitative difference in performance suggests that Hebbian plasticity provides a more "natural" mechanism of memory for the purpose of familiarity detection.

The generalization performance of HebbFF is due to the fact that the memory representation of an item does not change over time, other than being scaled by a factor. A stimulus $\mathbf{x}(t)$ is

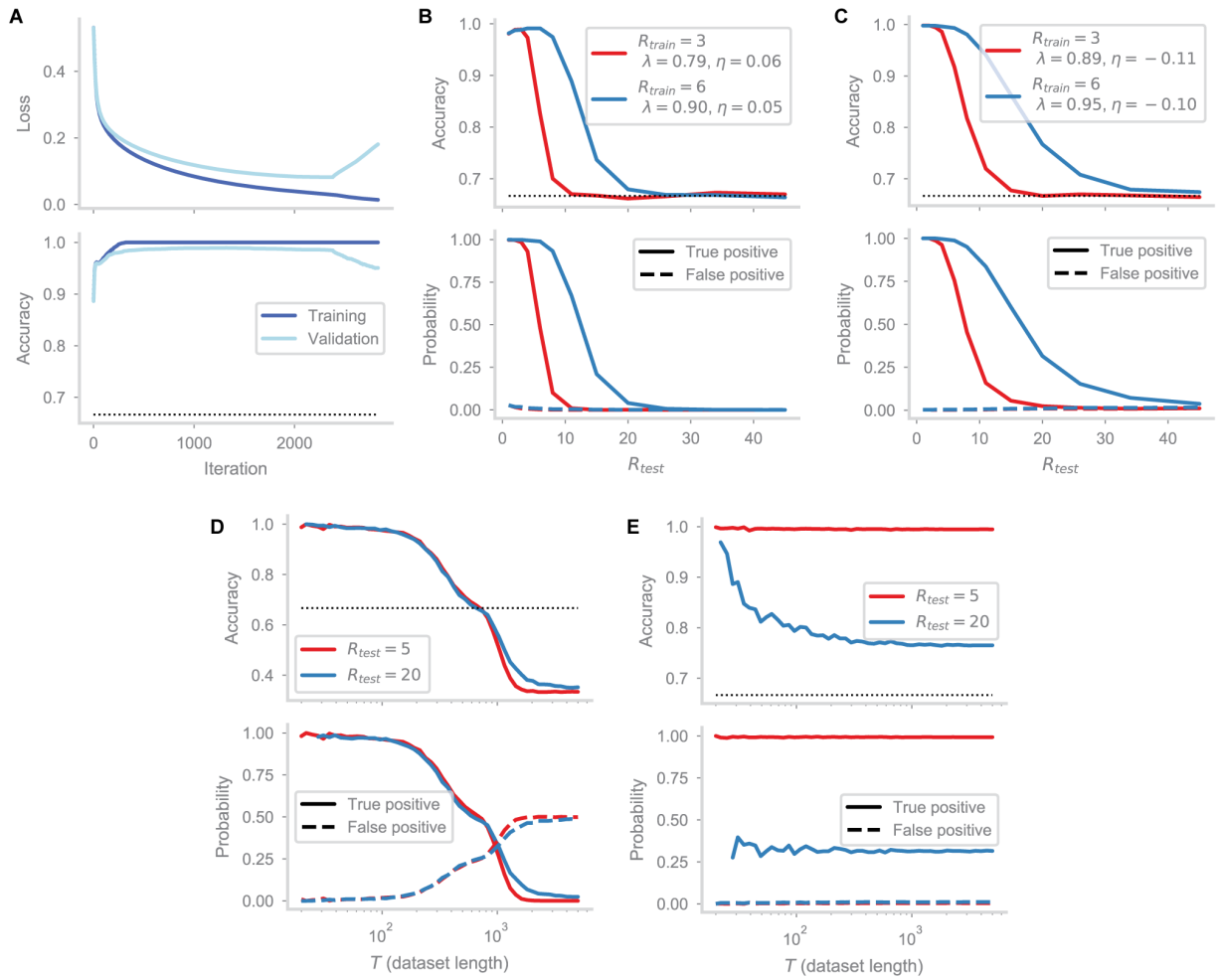


Figure 2.3: Hebbian vs. anti-Hebbian plasticity and continual operation. (A) Training the HebbFF network ($d = N = 100$), as in Figure 2.2A. Both training and validation loss decrease, and accuracy increases, for a single instance of the dataset with $R = 3$, indicating in-distribution generalization. Over many iterations, however, overtraining occurs due to the use of a single dataset, increasing the final validation loss. (B) HebbFF network trained with "infinite data" as in Figure 2.2B ($R = 3$, red; $R = 6$, blue) shows not only in-distribution generalization to any dataset with $R = 3$ ($R = 6$, resp.), but also out-of-distribution to datasets with any smaller R and some larger R's. (C) HebbFF with a different initialization converges to a qualitatively different solution with a negative learning rate η, an anti-Hebbian learning rule (see also Figure 2.15). The anti-Hebbian solution shows generalization performance over a larger range of R values than Hebbian. (D) Model from [161], evaluated on the continual familiarity detection task, varying the length T of the trial. Accuracy (top) is near-perfect regardless of the repeat interval R (blue vs. red curve) until the model reaches its capacity ($P^* \approx 100$ for network size $d = N = 100$) because the model reliably stores the first P^* patterns. Accuracy rapidly drops below chance for $T > P^*$ as the model begins to report familiar stimuli as novel (see Figure 2.9B). (E) HebbFF network operates continuously, as its accuracy (top) is consistent with the generalization curve from (C), with near-perfect performance for $R_{test} = 5$ and above 80% for $R_{test} = 20$ for any trial length. True and false probabilities (bottom) are better representations as accuracy (top) is artificially higher for small T due to the low proportion of familiar stimuli.

initially stored as the outer product of $\mathbf{h}(t)$ and $\mathbf{x}(t)$, multiplied by the plasticity rate η. The plastic component of the connectivity matrix also contains terms arising from previously stored memories which, for the purposes of this particular stimulus, act as additive noise ε:

$$\mathbf{A}(t+1) = \eta \mathbf{h}(t)\mathbf{x}(t)^T + \varepsilon$$
$$\mathbf{A}(t+k) = \lambda^k \eta \mathbf{h}(t)\mathbf{x}(t)^T + \lambda^k \varepsilon + \varepsilon'$$

Unlike HebbFF, RNNs are poor at generalizing across intervals R because the dynamics of their units allow the memory representation of a stimulus to change arbitrarily over time. The RNN only generates the appropriate representation at the time when a query is expected, namely after a delay equal to the value of R used during training. This makes it difficult to generalize across intervals.

2.2.4 The "how" of synaptic plasticity: storage via an anti-Hebbian rule

The plasticity rate η in HebbFF can be positive or negative, resulting in either Hebbian or anti-Hebbian plasticity. For the Hebbian solution with $\eta > 0$, synapses are potentiated in response to a stimulus. When it is repeated, the hidden layer activity is higher than for a novel stimulus due to the increased strength of the synapses storing the memory. For anti-Hebbian plasticity, $\eta < 0$, synapses are depressed when a memory is stored. In this case, the hidden layer activity is lower for a familiar stimulus than a novel stimulus, which is consistent with experimental results of repetition suppression [164, 165, 167]. Furthermore, the meta-learning algorithm is more likely to converge to the anti-Hebbian solution, especially when trained with a relatively large repeat interval, even if the initial value of η is positive, and almost always when the initial value is negative (Figure 2.15).

Interestingly, anti-Hebbian plasticity enables successful familiarity detection over considerably longer intervals than a Hebbian rule (Figure 2.3C). To understand this, note that the memory of a stimulus is degraded in two ways: additional plasticity events obscure existing memories, and the plastic weights decay over time. With an anti-Hebbian plasticity rule, the hidden layer activation $\mathbf{h}(t)$ is close to zero for a familiar stimulus due to repetition suppression. As a result, the plasticity

update $\eta \mathbf{h}(t)\mathbf{x}(t)^T$ when the stimulus is repeated is negligible – as if a stimulus was not presented at that time step. This effectively reduces the number of plasticity events, reducing the disruption of existing memories. As a secondary effect, the smaller number of plasticity events allows a larger λ (smaller decay rate) to be used while still controlling the amplitude of plastic weights (Figure 2.15). This slower decay rate further extends the lifetime of the memory. Due to their superior performance and consistency with experimental results, we only consider anti-Hebbian solutions throughout the following sections.

2.2.5 The "when" of synaptic plasticity: continual learning without catastrophic forgetting

Previous modeling work using anti-Hebbian plasticity mechanisms for familiarity detection [161] focused on a paradigm used in classic studies of recognition memory [160] in which subjects are serially presented an entire dataset and later asked to identify which stimulus is familiar in a two-alternative-forced-choice (2AFC) test. Analogously, this previous modeling work used explicit "learning" and "testing" phases and demonstrated an impressive capacity for recognition memory [161] (Figure 2.9A). When evaluated on the continual memory task that we use, the Bogacz-Brown model has near-perfect performance if the number of stimuli T in the dataset is smaller than the model's capacity P^*, independent of the value of the repeat interval R (Figure 2.3D). That is, the model successfully stores all $T < P^*$ stimuli. As the dataset size T increases, however, the model performance declines due to catastrophic interference (Figure 2.3D, Figure 2.9B; section 2.4). To store additional memories, the old memories must be removed by resetting the synaptic weights.

In real-world scenarios, however, an organism typically does not experience a dedicated "learning" phase. The answer to "when" synaptic plasticity should occur is "always." As such, the HebbFF model operates continually rather than using separate learning and evaluation phases. Its performance is independent of the length of the dataset, and it can operate continuously without any need to reset the synaptic weights. For example, a HebbFF network trained with $R = 5$ operates at near-perfect performance irrespective of the duration of the trial T when tested with $R = 5$ (Figure 2.3E). Similarly, when tested with $R = 20$, it operates continually at near 80% accuracy

(Figure 2.3E), as expected from the generalization curve in Figure 2.3C (note that for small T the accuracy (Figure 2.3E, top, blue) is transiently elevated because the fraction of novel stimuli is more than $\frac{2}{3}$). In other words, the model has a moving window in time within which it can successfully detect a familiar stimulus, and it forgets old stimuli gracefully without suffering from catastrophic interference.

2.2.6 The "where" of synaptic plasticity: addressing via strong feedforward weights

In the HebbFF network, the hidden layer plays a dual role. On the one hand, it must produce a reliable familiarity signal for the readout to decode. On the other, it must create a robust representation of the input stimulus during the Hebbian plasticity update. The hidden activity is controlled by the fixed parameters $\mathbf{W_1}$ and $\mathbf{b_1}$, as well as the plastic matrix $\mathbf{A}(t)$. Here, we investigate how $\mathbf{W_1}$, $\mathbf{b_1}$, and $\mathbf{A}(t)$, impact these two aspects of the familiarity detection task.

To simplify this analysis, we restrict $\mathbf{W_2}$ to be a scaled 1-by-N matrix of ones, $\mathbf{W_2} = \alpha_2[1, \ldots, 1]$, where α_2 is a trained scalar. Similarly, we restrict $\mathbf{b_1} = \beta_1[1, \ldots, 1]^T$. Since the hidden units now contribute equally to the readout, they are statistically identical (although not necessarily independent). Therefore, the rows of $\mathbf{W_1}$ and $\mathbf{A}(t)$ are statistically identical, allowing us to meaningfully plot histograms of the corresponding input currents. Empirically, performance is not affected by this choice of output weights (Figure 2.10A), the distribution of $\widehat{y}(t)$ for familiar and novel stimuli is the same (Figure 2.10B), the readout vector $\mathbf{W_2}$ and the bias term $\mathbf{b_1}$ have similar features (Figure 2.10C), and anti-Hebbian plasticity is still the preferred form of plasticity.

Networks trained with larger R have sparser hidden unit activity (Figure 2.3A-C): the sparser the activity, the less plasticity is evoked, and thus the longer memories can be retained without overwriting. In the limiting case we might expect that exactly one neuron is active for a novel stimulus and none are active for familiar stimuli. Associated with this increased sparsity in activity, $\mathbf{W_1}$ is also sparser for larger R (Figure 2.3D-F, Figure 2.15).

To isolate the effect of $\mathbf{W_1}$ on hidden unit activity, we compute a histogram of the input current into the hidden layer due to the non-plastic synapses, $\mathbf{W_1 x}(t) + \mathbf{b_1}$, across units and across time

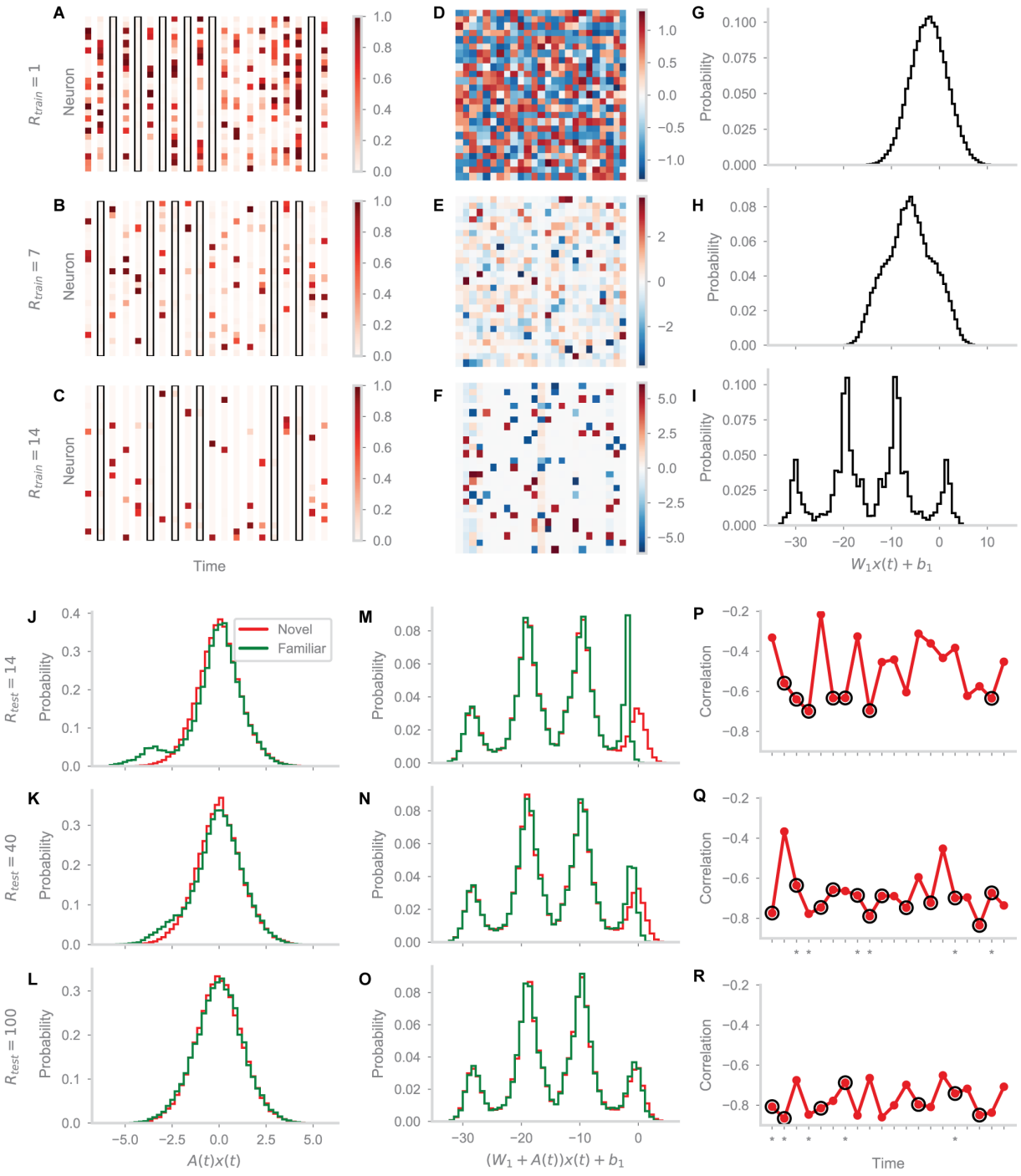

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
$R_{train} = 1$
$R_{train} = 7$
$R_{train} = 14$
$R_{test} = 14$
$R_{test} = 40$
$R_{test} = 100$
Neuron
Time
Probability
Novel
Familiar
Correlation
$W_1x(t) + b_1$
$A(t)x(t)$
$(W_1 + A(t))x(t) + b_1$

Figure 2.3 *(previous page)*: Storage and readout mechanism. (A-C) Hidden layer activity $\mathbf{h}(t)$ over 20 consecutive timesteps for networks with input dimension $d = 25$ and $N = 25$ hidden units, trained on datasets with $R = 1,\ 7,$ or 14, respectively. Familiar stimuli (black rectangles) cause silencing i.e. repetition suppression of hidden layer activity. Activity for novel stimuli becomes sparser for networks trained with larger R. (D-F) Static weight matrix $\mathbf{W_1}$ of the networks from (A-C). The weight matrix becomes sparser (Figure 2.15), and individual weight magnitudes increase for networks trained with larger R, enabling more sparse activity in the hidden layer for novel stimuli. (G-I) Distributions of input current into the hidden layer due to the static component of the synapses, i.e. the matrix $\mathbf{W_1}$ and bias $\mathbf{b_1}$, for the networks from (A-C). We do not distinguish familiar and novel stimuli since the current due to the static component is the same regardless of novelty. For networks trained with larger R, the distribution becomes multi-modal, with the number of modes equal (approximately) to the number of high-magnitude values per row of $\mathbf{W_1}$, plus one. Due to the bias, only the rightmost mode has the potential to produce firing rates that are significantly above zero. (J-L) Distributions of input current into the hidden layer due to the plastic component of the synapses, i.e. the matrix $\mathbf{A}(t)$, for novel (red) and familiar (green) stimuli. We only consider the trained network from (C,F,I) and evaluate its behavior on test sets with $R = 14,\ 40,$ or 100, corresponding to perfect, intermediate, and chance accuracy. The large central mode occurs due to stored stimuli uncorrelated with the input stimulus $\mathbf{x}(t)$. In the novel case, the input is uncorrelated with all the stored stimuli by definition, and thus there is only one mode. Similarly, in the familiar case with a long delay interval $R = 100$ the stored stimulus has decayed sufficiently that its signal is lost. In the case of familiar stimuli presented at shorter delay intervals, $R = 14$ or 40, there is an additional mode due to the correlation between the input $\mathbf{x}(t)$ and its copy $\mathbf{x}(t - R)$ previously stored in the plastic matrix $\mathbf{A}(t)$. (M-O) Distributions of the total input current into the hidden layer on test sets with $R = 14,\ 40,$ or 100. Only the values above zero cause high firing rates after applying the logistic sigmoid nonlinearity. Since all the input currents are low for familiar stimuli (green) for small values of R, there is repetition suppression. (P-R) Correlation between the input current into the hidden layer from static and plastic synapse components at each of 20 consecutive timepoints. Asterisks indicate output response errors. For sufficiently small R, the input currents are more anti-correlated for familiar stimuli (black circles) than for novel. Combined with the distributions of input currents, this indicates that the units receiving positive input current from the static synapses receive negative input current from the plastic synapses.

(Figure 2.3G-I). As R increases, the distribution becomes multi-modal as a result of the combinatorial structure of the rows in $\mathbf{W_1}$ (more evident in the idealized model: see below and section 2.4). In general, the number of peaks in this distribution depends on the number of large-magnitude values of $\mathbf{W_1}$ per row. Critically, due to the logistic function nonlinearity, only the rightmost peak in Figure 2.3I is large enough to elicit appreciable activity in the hidden layer. This peak drives the small number of hidden units that are significantly activated by a novel stimulus. In other words the $\mathbf{W_1}$ matrix acts like an addressing function to select a small subset of hidden units to store the memory of a given stimulus.

We next consider the effect of $\mathbf{A}(t)$, focusing on the network trained to maximum capacity (Figure 2.3C,F,I) (see next section). For a novel stimulus, the distribution of the input current due to the plastic synapses $\mathbf{A}(t)\mathbf{x}(t)$ is unimodal and symmetric about zero (Figure 2.3J-L). For a familiar stimulus, however, there is an additional peak at approximately $\lambda^{R-1}\eta d$. This peak is due to the dot product of the input vector $\mathbf{x}(t-R)$ (stored in the matrix $\mathbf{A}(t)$ as $\lambda^{R-1}\eta\mathbf{h}(t-R)\mathbf{x}(t-R)^T$), and the familiar input vector $\mathbf{x}(t) = \mathbf{x}(t-R)$. Importantly, the neurons that exhibit this behavior are the same ones active due to $\mathbf{W_1}$ when the stimulus was novel. Thus, again, $\mathbf{W_1}$ provides addressing functionality (now indirectly through its effect on $\mathbf{A}(t)$), allowing the system to probe the same neurons not only during storage but also during recall.

Finally, the total hidden layer input current is the sum of these two components, $(\mathbf{W_1} + \mathbf{A}(t))\mathbf{x}(t) + \mathbf{b_1}$ (Figure 2.3M-O). Comparing Figure 2.3I and Figure 2.3O, we see that the large central symmetric mode of the $\mathbf{A}(t)\mathbf{x}(t)$ distribution does not significantly impact the total hidden layer input current. Rather, the familiarity signal arises because the smaller peak of the $\mathbf{A}(t)\mathbf{x}(t)$ distribution pushes the rightmost peak of the $\mathbf{W_1}\mathbf{x}(t) + \mathbf{b_1}$ distribution below zero (Figure 2.3M). Anti-correlation between the two input currents for familiar stimuli (Figure 2.3P-R) indicates that this shift is caused by the input current from the plastic component of the synapse cancelling the input current from the fixed component, resulting in lower activation, i.e. repetition suppression.

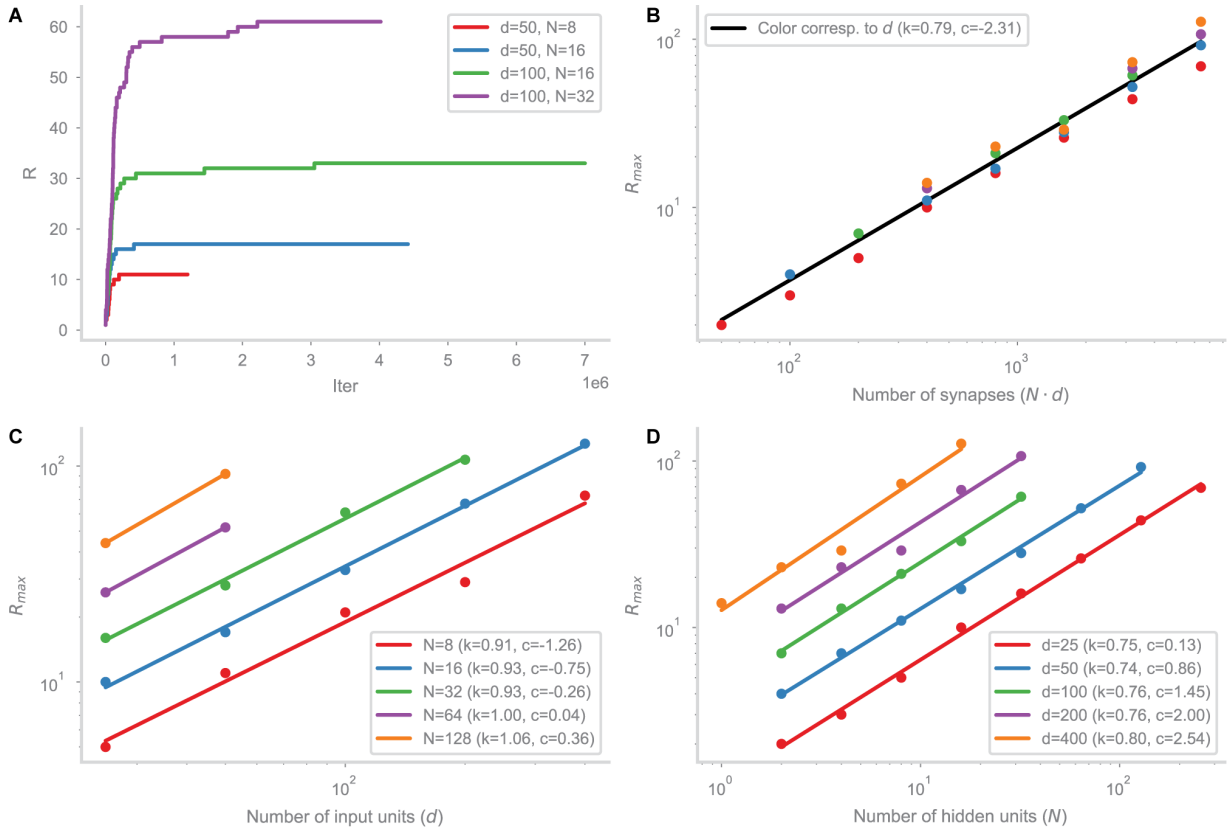


Figure 2.4: Curriculum training and empirical capacity. (A) The value of R used over the course of curriculum training for four different network sizes. R is incremented once the network achieves an accuracy > 0.99. Training is considered converged when the value of R is not incremented for at least 1 million iterations. (B) The final value of R after curriculum training (i.e. network capacity) as a function of the number of plastic synapses in the network, plotted on a log-log scale. The color of the points corresponds to the number of input units, colors from panel (D). The least-squares fit (black line, slope k, bias c) indicates that the empirical network capacity scales sub-linearly with the number of synapses. (C) Capacity as a function of the input dimension d for various hidden layer sizes N. (D) Capacity as a function of the hidden layer size for various input dimensions d. Capacity primarily depends only on the number of synapses, rather than on the hidden or input layer sizes.

2.2.7 Curriculum training and empirical capacity

A randomly-initialized HebbFF network may fail to find a solution if directly trained with a large value of R (Figure 2.15). Instead, we use a curriculum training procedure to bootstrap the optimized solution. First, the network is trained on data with $R = 1$, using the "infinite data" regime. Once the accuracy is above 99%, R is incremented by one and training continues on data with $R = 2$. This process continues until R becomes large enough that the network cannot find a solution with accuracy above 99%, i.e. if R is not incremented for at least 2 million iterations (Figure 2.4A). We thus define the memory capacity $R_{\max}$ as the largest value of R for which the familiarity detection accuracy is above 99%.

We curriculum-train networks of different sizes and plot the capacity $R_{\max}$ for each one (Figure 2.4B). For consistency and ease of training, we restrict the networks to the anti-Hebbian solution and use the uniform readout as above. We find that the capacity depends primarily on the number of synapses, rather than on the number of pre- or postsynaptic neurons (Figure 2.4C,D), consistent with previous familiarity detection results [161]. To estimate the scaling, we compute a linear least-squares fit of $\log(R_{\max})$ as a function of $\log(Nd)$. Empirically, we find that the capacity of the network scales as

$$R_{\max} \approx 0.10(Nd)^{0.79}$$

In contrast, the model of Bogacz and Brown [161] for the non-continual task has a capacity that is linear in the number of synapses. To determine whether the difference between the empirical performance of HebbFF and the Bogacz-Brown model reflects a fundamental limitation in the feedforward architecture, we developed an idealized version of the model (Figure 2.5A) that we could study analytically (section 2.4).

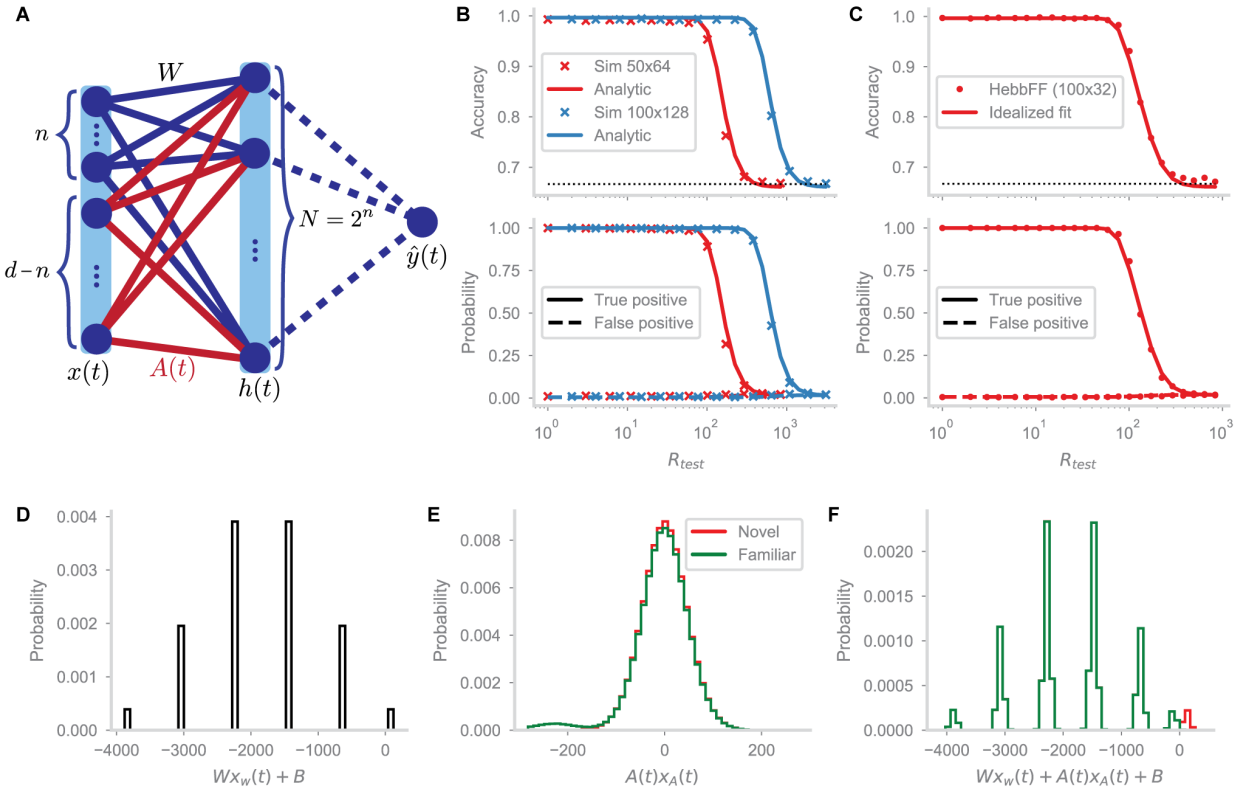


Figure 2.5: Idealized model. (A) The idealized HebbFF network architecture. In contrast to the original HebbFF network with a single effective matrix $\mathbf{W_1}+\mathbf{A}(t)$, the input $\mathbf{x}(t)$ is effectively split into two sections of size n and $D = d - n$ that serve as inputs into separate static and plastic synaptic matrices $\mathbf{W_1}$ and $\mathbf{A}(t)$, respectively (section 2.4). The hidden layer size is $N = 2^n$. The readout unit outputs $\widehat{y}(t) = 1$ whenever any of the hidden units is active. (B) The analytic calculation of network performance (solid line) matches simulation results for the idealized network (x's), shown for two different network sizes (red, blue). (C) A least-squares fit of the analytic performance curve of the idealized network to a trained HebbFF network of the same size for two network sizes. The idealized network has similar performance to the HebbFF model if its decay rate and bias are scaled appropriately: $\lambda \approx 0.986,\ b_1 \approx -4.771$ (for all units) for $d = 200,\ N = 32$, and $\lambda \approx 0.993,\ b_1 \approx -4.771$ for $d = 200,\ N = 32$. (D-F) Same as Figure 2.3(L,J,M), but for the idealized network ($D = 400,\ N = 32,\ R = 300$).

2.2.8 Idealized model and theoretical capacity

We noted above that the limiting behavior of the network at maximum capacity appears to have $\mathbf{W_1}$ activate just a single unit for memory storage. We build this limiting behavior into the idealized model through a specific choice of $\mathbf{W_1}$ and $\mathbf{b_1}$, set by design rather than through a training procedure. Specifically, we use the first $n \ll d$ components of $\mathbf{x}(t)$ as an identifier by choosing the first n columns of $\mathbf{W_1}$ so that a unique hidden unit is activated by each possible n-bit combination of these components (section 2.4), and set the remaining columns of $\mathbf{W_1}$ to zero. To simplify the model, we do not allow plasticity to operate on the inputs from these bits and set the first n columns of $\mathbf{A}(t)$ to zero (Figure 2.5A). This isolates the addressing function of the fixed matrix from the memory storage. Furthermore, instead of a sigmoid nonlinearity for the hidden units, we use a Heaviside step function $\Theta(\cdot)$. Thus, the hidden layer in the idealized model is governed by

$$\mathbf{h}(t) \;=\; \Theta\left(\left(\mathbf{W_1} + \mathbf{A}(t)\right)\mathbf{x}(t) \;+\; \mathbf{b_1}\right)$$

For the nonzero entries of $\mathbf{A}(t)$, plasticity is the same as in the trained model. However, because the Heaviside function does not depend on the scale of the input, we can set the plasticity rate to $\eta = -1$ without loss of generality. The optimal synaptic decay rate λ can be computed analytically. Finally, a stimulus is considered familiar if all hidden unit activities are identically zero, and novel otherwise (section 2.4).

This idealized model exhibits qualitatively similar behavior to HebbFF. We can fit the analytic functional form of the true and false positive probabilities computed from the idealized model (Figure 2.5B) to the corresponding probabilities of HebbFF (Figure 2.5C). Furthermore, the histograms of inputs to the hidden layer are qualitatively similar: $\mathbf{W_1}\mathbf{x}(t) + \mathbf{b_1}$ has the same multimodal distribution with more prominent peaks in the middle (Figure 2.3I, Figure 2.5D) (due to the structure of $\mathbf{W_1}$, see section 2.4), a bimodal distribution of $\mathbf{A}(t)\mathbf{x}(t)$ with a large symmetric central peak and a smaller one corresponding to the familiarity signal (Figure 2.3J, 6E), and a similar distribution of the total input current $(\mathbf{W_1}+\mathbf{A}(t))\mathbf{x}(t) + \mathbf{b_1}$ (Figure 2.3O, Figure 2.5F). From this,

we conclude that the memory storage and readout mechanisms are analogous in the meta-learned HebbFF network and the idealized model.

Along with the true and false positive probabilities, the memory capacity of the idealized model can be computed analytically (section 2.4). As in the Bogacz-Brown model [161], the capacity, as characterized by 99% accuracy, is proportional to the number of synapses Nd. There are several possible reasons for the discrepancy between this analytic capacity, as well as that of the Bogacz-Brown model, relative to the empirical capacity for HebbFF.

First, the idealized HebbFF model uses a dedicated set of synapses through the fixed $\mathbf{W_1}$ matrix, and the Bogacz and Brown model selects the units that have the highest input current implicitly through inhibitory competition. Both of these are dedicated addressing functions for the hidden layer, but meta-learned HebbFF must multiplex this functionality with memory storage, leading to correlations between the hidden layer input currents from the plastic and fixed synapse components (Figure 2.11A).

In addition, replacing the logistic function with a Heaviside function means that familiar stimuli truly generate no plasticity in the idealized model, reducing overwriting at the cost of not reinforcing partially-decayed memories (Figure 2.11B-C). For the same reason, in contrast to HebbFF, the idealized model achieves maximal plasticity for any suprathreshold level of input to a hidden layer unit.

Finally, training the HebbFF model may lead to specialized solutions for small d and N that have better performance than that predicted by the asymptotic analysis. Similarly, training may not converge to the optimal solution for large d and N because it requires the use of very long repeat intervals R. This means the dataset size T must be very large to include a sufficient number of familiar examples, which may lead to practical issues such as vanishing gradients. Thus, the empirical capacity may scale sublinearly with the number of synapses because of over-performance at low R, under-performance at high R, or both.

2.2.9 HebbFF recapitulates neural data from inferotemporal cortex

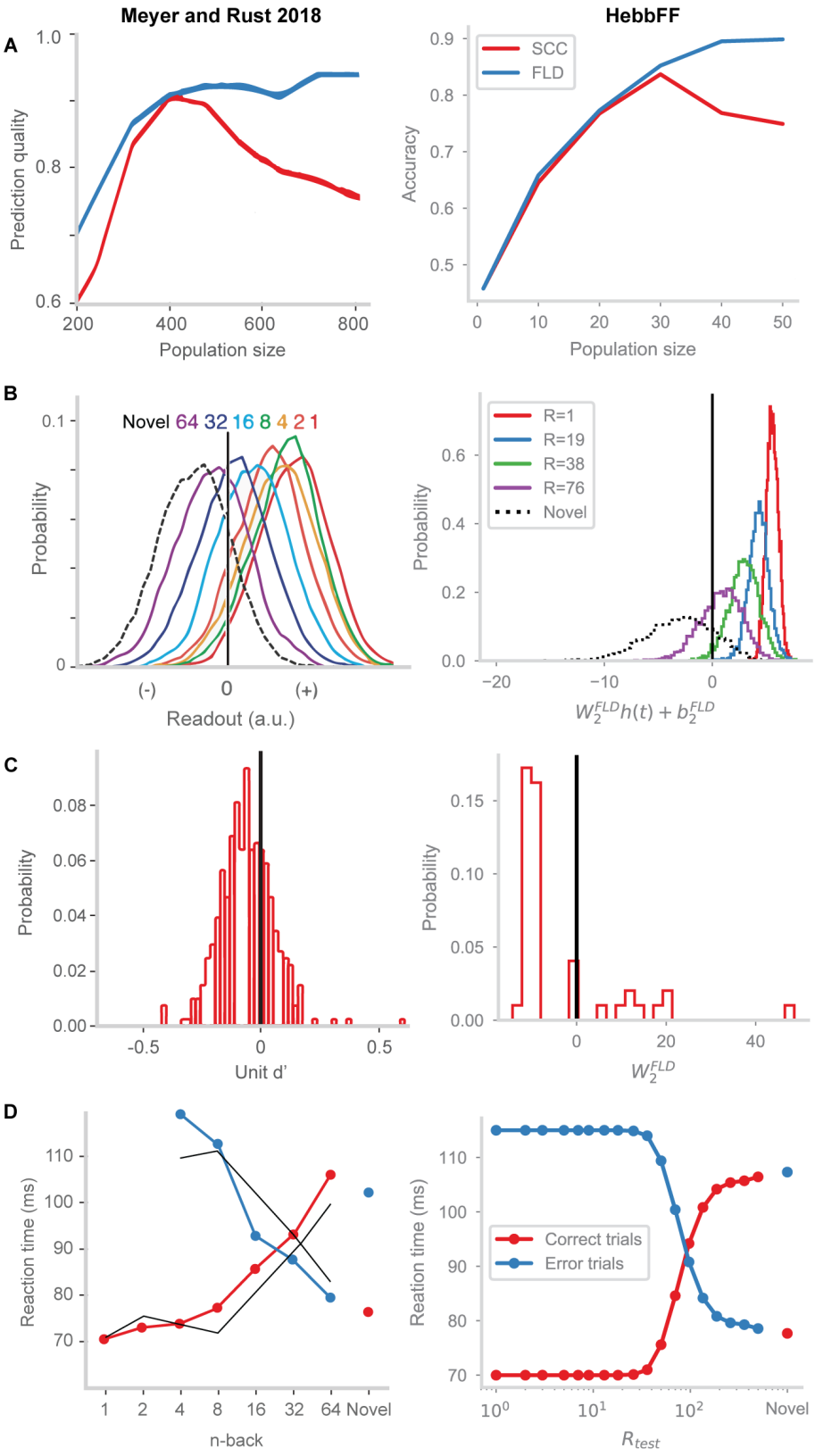

A
Meyer and Rust 2018
HebbFF
Prediction quality
Population size
Accuracy
SCC
FLD
B
Novel 64 32 16 8 4 2 1
Probability
Readout (a.u.)
R=1
R=19
R=38
R=76
Novel
C
Unit d'
D
Reaction time (ms)
n-back
Correct trials
Error trials
Reation time (ms)

Figure 2.6 *(previous page)*: Comparison to IT cortex data. (A) Left: neurons from the IT cortex used to predict the behavioral outputs of a monkey performing continual familiarity detection, decoded using the Fisher linear discriminant (FLD, blue) or spike count classifier (SCC, red). Right: units from the hidden layer of a trained HebbFF network (trained with unconstrained $\mathbf{W_2}$) used to detect familiarity obtained from SCC and FLD decoders. In both cases, the number of neurons/units available to the decoder was varied, added in order of increasing weight according to the FLD decoder. While the FLD decoder accuracy saturates, the SCC decoder accuracy peaks and begins to decline as more neurons/units are included in the decoder. (B) Distribution the FLD decoder output for IT cortex neurons (left) and HebbFF hidden units (right) for novel stimuli, and familiar stimuli at varying delay intervals. In both cases, the distribution shifts towards lower values as delay interval increases. For HebbFF, the distribution gets narrower for shorter delay intervals due to saturation in the hidden layer units. (C) Distribution of the FLD decoder weights for decoding IT cortex data (left) or HebbFF hidden unit activity (right). In both cases, the majority of output weights are negative, with some positive values. (D) Left: measured reaction time as a function of delay interval for correct and error trials (red, blue curves) in monkeys performing the continual familiarity detection task. Black lines indicate reaction times predicted using strength theory analysis. Right: HebbFF predicted reaction times using analogous strength theory analysis, using constants of proportionality from [165] (section 2.4). Both result in a qualitatively similar x-shaped pattern. Plots on the left side of (A-D) adapted from [165]. (E) The weight matrix $\mathbf{W_1}$ of the HebbFF network trained on the augmented task, requiring simultaneous classification and familiarity detection. Hidden units split into "classification" and "familiarity" units, with classification units (marked with asterisks) having very strong input weights to overcome the noise from the plastic $\mathbf{A}(t)$ matrix.

We next compare the optimized HebbFF model with experimental results. Meyer and Rust (2018) recorded neurons from the inferotemporal (IT) cortex of monkeys performing familiarity detection and compared the quality of two decoders in predicting behavior from neural data as a function of neural population size. The authors considered a "spike count classifier" (SCC) decoder, which amounts to comparing a simple average of neuronal firing rates to a threshold, as well as a Fisher linear discriminant (FLD), which instead considers a weighted average, with weights computed from the data [165].

We perform a similar analysis. We first construct an FLD decoder of the hidden unit firing rates and rank the units in reverse order of their FLD readout weights (i.e. units with the most negative weights are top-ranked; section 2.4). We then consider decoders that use increasingly larger subsets of hidden units, adding them according to their ranking [165]. As in the experimental data, performance saturates for the FLD and declines for the SCC readout beyond a certain number of

decoded units (Figure 2.6A). This can be explained by the fact that some units do not provide a reliable signals of familiarity – in the network this is due to suboptimal training, and in the IT cortex possibly due to those neurons performing an unrelated task (see supplement). Including them hurts performance of the SCC decoder, but since the FLD readout weight for these units is close to zero, they do not alter its familiarity detection performance.

Comparing the experimental and model distributions of readout activity shows a qualitatively similar pattern for outputs to novel and familiar stimuli (Figure 2.6B). Both distributions shift toward smaller values as R increases, as the outputs for familiar stimuli approach those for novel. The fact that the distribution of outputs becomes narrower for HebbFF as R decreases, unlike in the data, may be due to repetition suppression causing hidden units to have near-zero responses for highly familiar (low R) stimuli, thus causing the readout distribution to cluster around its minimal value. On the other hand, biological neurons that exhibit repetition suppression may never be fully silenced – for example if it takes multiple repetitions to achieve maximal familiarity or if neurons are multiplexed with another task that requires a baseline level of activity. Furthermore, as in the data, the distribution of readout weights is biased towards negative values (Figure 2.6C).

Finally, using a similar "strength theory" analysis as in the experimental results [165, 179], which suggests that reaction times are inversely proportional to the distance of the readout from the threshold, we can qualitatively reproduce the x-shaped reaction time curves seen in the data [165]. We used the same proportionality constant determined experimentally to compute network "reaction times" (Figure 2.6D). Overall, we find that the HebbFF model captures a number of features seen in the experimental results.

2.2.10 Two subpopulations emerge in a classification-augmented task

IT cortex encodes object identity as well as familiarity [180, 181]. To match this dual functionality, we augment familiarity detection with object classification. We first create a large pool of random vectors and randomly assign a binary label to each one. We then generate a familiarity detection dataset as before, except that each novel input is drawn from this pool (without replacement) rather

than being generated anew. In addition to the scalar readout of familiarity, the network must now report the class of the stimulus through a second binary output. Critically, both outputs are read out from the same hidden layer activity (section 2.4).

The augmented task could be solved by having all the neurons multiplexed to encode both familiarity and object identity. Alternatively, the neurons could split into two subpopulations, one of which detects familiarity and the other classifies objects [182]. We find that the HebbFF model converges to this second solution, an even split between familiarity and classifier units, as evident from inspecting the $\mathbf{W_1}$ matrix (Figure 2.6E). Consistent with this, the capacity of the classifier-augmented HebbFF with 50 hidden units ($R_{\max} \approx 13$) is approximately the same as the original network with 25 units ($R_{\max} \approx 14$). In accord with this split, SCC decoder performance peaks in the split-task network when half of the top-ranked units are included (Figure 2.12D) because including units responsible for object identity but not familiarity degrades the familiarity readout. The other similarities to experimental results discussed in the previous section also hold for the task-augmented network (Figure 2.12).

2.2.11 Familiarity detection of real images

To validate the HebbFF model in a more realistic scenario, we evaluate its performance on real-world object images. We consider the dataset used by Brady et al. to study familiarity detection in humans [162]. As a stand-in for the processing done by the visual stream before the inferotemporal or perirhinal cortices, we use a pre-trained convolutional neural network (CNN), and extract the activity in its penultimate layer (before the final classification step). We use the ResNet18 network [90], although any CNN could, in principle, be used. This activity is a 512-dimensional vector, which, if used as the HebbFF input dimension d, would lead to the capacity $R_{\max}$ being prohibitively large for training purposes. To keep the performance in a reasonable range, we downsample to $d = 50$, either by partial sampling (Figure 2.7A) or by introducing an intermediate layer (Figure 2.13A). We use the uniform readout $\mathbf{W_2}$ for simplicity of training and analysis, although the results are similar for the unconstrained readout.

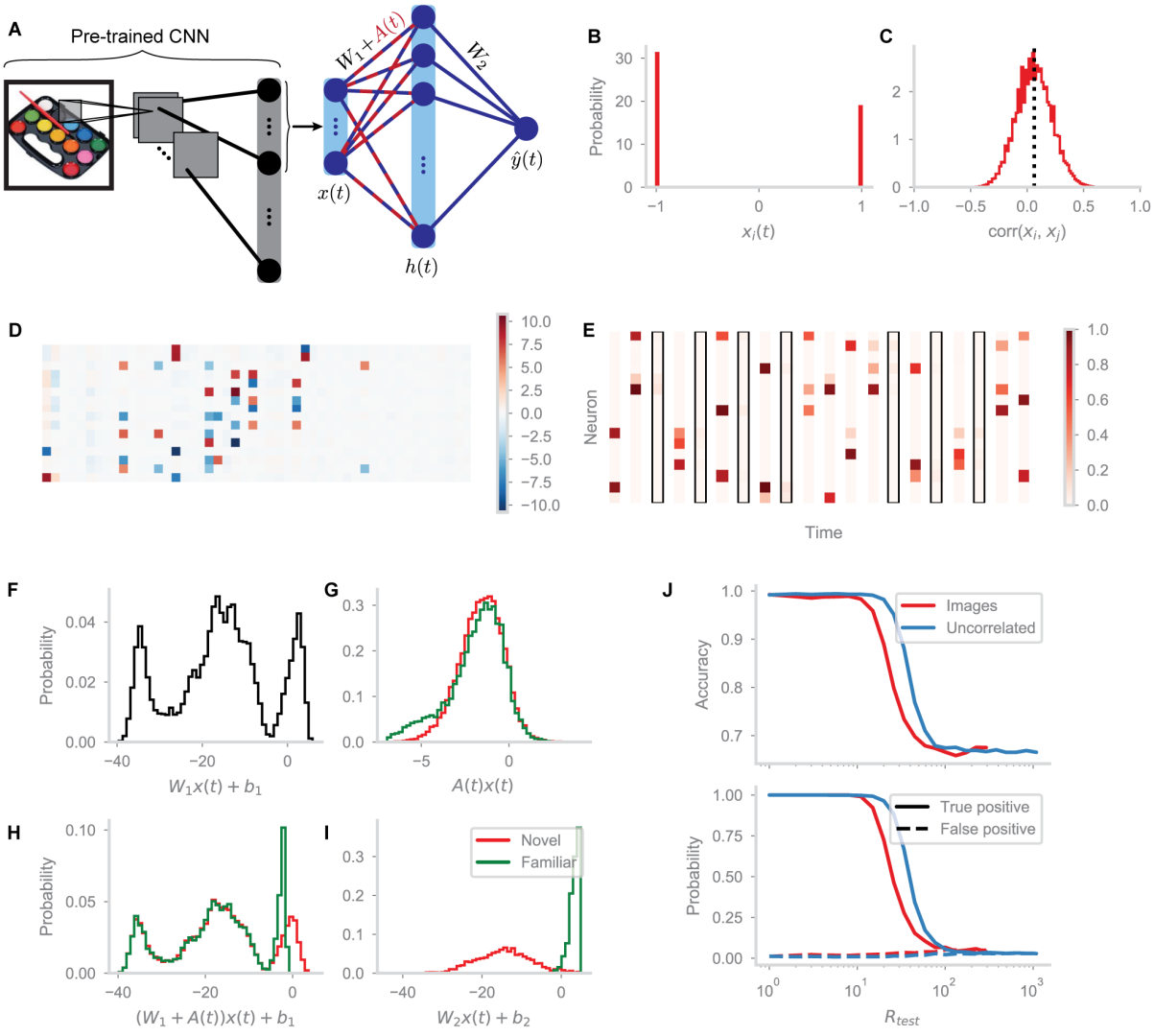


Figure 2.7: HebbFF performance on real-world images. (A) Network architecture for familiarity detection of real-world images. The activity of the penultimate layer of a convolutional neural network (ResNet18, pre-trained on ImageNet) is downsampled and passed to the HebbFF network ($d = 50,\ N = 16$) for familiarity detection. Only the HebbFF portion of this network is trained, via curriculum training. (B) Distribution of inputs $\mathbf{x}(t)$ to HebbFF. After down-sampling by extracting the first 50 units of the CNN, the activity is centered at zero and binarized. (C) Histogram of the correlations between all pairs of input stimuli $\mathbf{x}(t)$. On average (vertical dashed line) the correlation is slightly positive. (D-H) Same plots as Figure 2.3(F,C,I,J,M), respectively ($R_{train} = R_{test} = 12$). (J) Generalization performance, compared to a network of the same size trained on uncorrelated binary random vectors, is lower due to correlations in the input images.

As the first method of downsampling, we truncate the output of the CNN (Figure 2.7A). To keep the same input datatype as in previous sections, we also shift the inputs to zero mean and binarize them by taking the sign of each input component (Figure 2.7B). Unlike in previous sections, however, the inputs to HebbFF now have correlations that tend to be positive (Figure 2.7C). Nevertheless, this network has qualitatively similar features as the networks trained on uncorrelated vectors. The $\mathbf{W_1}$ matrix has a similar structure (Figure 2.3F, 8D), the hidden layer activity is sparse (Figure 2.3C, 8E), and the hidden unit input current distributions have similar shapes (Figure 2.3I,J,M, 7B, 8F-I). Due to the added correlations, however, there is a decline in performance compared to a network of the same size trained and evaluated on uncorrelated binary random vectors (Figure 2.7J).

As another way to downsample, we add a trainable linear layer that transforms the CNN output to a 50-dimensional real-valued vector (Figure 2.13A). After training, the resulting inputs to HebbFF are no longer binary, but they are zero-mean (Figure 2.13B) and have zero-mean correlations Figure 2.13C). Interestingly, the network learns to generate this representation automatically to optimize familiarity detection over long intervals, which further supports storing uncorrelated stimuli. Although the $\mathbf{W_1}$ matrix (Figure 2.13D) and the distribution of input currents from the fixed component of the synapses (Figure 2.13F) have a different structure compared to the original network, the operating principle remains the same: the $\mathbf{W_1}$ matrix acts as a addressing function to select a unique neuron in the hidden layer (Figure 2.13E) that is then suppressed for a familiar stimulus through the $\mathbf{A}(t)$ matrix (Figure 2.13G-H). The network maintains its generalization performance across repeat intervals R, and across permutations of the sequence of images (Figure 2.13J). However, it does not generalize well to images it has not been trained on. It is possible that this difficulty is due to the relatively small number of images used during training and may be addressed by using a much larger dataset such as ImageNet [183].

2.3 Discussion

Continual familiarity detection is a memory task that we perform every day, typically without being aware that we are doing it. We have used meta-learning to generate networks that solve this

task using synaptic plasticity. This is distinct from memory storage in RNNs that maintain memory traces through persistent activity. Given the extraordinary capacity and robustness of recognition memory, the idea that biological networks use ongoing activity for this purpose appears untenable [184, 185]. If a neuronal network is storing a stimulus by maintaining a particular firing rate pattern across its neurons, any other computation risks disrupting that memory trace. In contrast, storage through synaptic updates leaves the neuronal activity free to perform other computations unrelated to memory storage [131]. In addition, we found that synaptic plasticity provides a better inductive bias than recurrence for familiarity detection. After optimization through meta-learning, the HebbFF network not only outperforms RNNs on the task, but also easily generalizes both in-distribution and out-of-distribution of the training data. Although RNNs are a common approach to tasks that require storage of the input history [133, 91, 156], this result highlights the importance of considering alternative architectures and storage mechanisms, such as those that rely on synaptic, rather than neuronal, dynamics.

In answer to the question of "how" memories are stored, we find that anti-Hebbian plasticity, in which neuronal co-activation causes synaptic depression (alternatively this may be interpreted as potentiation of inhibitory synapses [186]), is a better storage mechanism for familiarity detection than Hebbian. An anti-Hebbian rule generalizes better, has a larger capacity, and is discovered by meta-learning more frequently and reliably. Although this result is consistent with previous work [161], the underlying reasons are different. Bogacz and Brown showed that in a non-continual version of the familiarity detection task, an anti-Hebbian plasticity rule leads to a larger storage capacity, although this advantage only held in the case of correlated inputs. In their case, the anti-Hebbian rule automatically suppresses common input features, effectively storing only the uncorrelated components, leading to an increased capacity. In contrast, anti-Hebbian HebbFF shows an advantage even for uncorrelated inputs in the continual task. This is due to an effective decrease in the number of plasticity events – a synaptic update is weak for a familiar stimulus because the postsynaptic activity is low, leading to smaller updates that are less disruptive to stored memories.

Equally important is the question of "where" memories are stored. HebbFF explicitly selects

storage locations through an addressing function implemented by strong feedforward weights $\mathbf{W_1}$, independent of the previously stored memories $\mathbf{A}(t)$. By inducing hidden layer activity – typically a single active neuron – $\mathbf{W_1}$ selects only those afferents for storing a novel memory. This is in contrast to implicit addressing through recurrent inhibition in a previous anti-Hebbian model [161] which selects 50% of hidden layer neurons. Although much experimental and theoretical work has been devoted to elucidating the plasticity rules used in memory storage, our work highlights the equal importance of studying the addressing functions of neuronal circuits as well.

Critically, unlike classical models, these answers emerged from meta-learning. The architectural features were not due to decisions made by the modeler but rather discovered through optimization. Although our particular meta-learning algorithm, BPTT, does not easily map onto a biological mechanism, we can nevertheless interpret it as a stand-in for structural changes over long time scales – an addressing function developing in a newborn's brain over the first years of her life, or a plasticity rule emerging within a species across generations. Evolutionary strategies for meta-learning [176, 173, 178] imply the latter interpretation. In contrast, the plasticity rule itself is a biologically realistic mechanism for learning over short time scales – seconds or minutes to store a memory that may be retrieved throughout a lifetime.

Thus, the HebbFF model predicts that there should be two populations of synapses – a small set of slow-varying or fixed synapses for addressing the memory neurons (the hidden layer of HebbFF) and a larger set of highly plastic synapses for encoding memories.

The HebbFF model also allows us to make a more quantitative experimental prediction. Although is obvious that the true positive rate (probability of correctly identifying a repeated stimulus as "familiar") should decrease with longer delay intervals R, we also observe that the false positive rate slightly increases (Figure 2.14A). This occurs because false negatives (incorrectly reporting a familiar stimulus as "novel") due to spuriously elevated activity in the hidden layer cause storage of a familiar stimulus in additional neurons, as determined by the addressing function. Subsequently, a novel stimulus is more likely to generate a false positive response due to a collision in the addressing function. (Note that although this effect increases false positives, it may also boost the true positive

rate since the memory gets more robustly encoded). Prior experimental paradigms have not measured this effect because each trial had familiar stimuli interleaved at various delay intervals, so novel stimuli could not be separated scored depending on the difficulty of the dataset and false positive probability was reported in aggregate. If biological networks implement familiarity detection through an anti-Hebbian plasticity mechanism, we expect the false positive rate to increase with larger R. Neither the Hebbian mechanism nor the RNN trained on a single R show this behavior (Figure 2.14B,C). Note, however, that anti-Hebbian plasticity is merely sufficient, not necessary, for this result, so the converse may not be true (Figure 2.14D).

There are experimental results that the HebbFF model does not capture. For example, data from human subjects shows a very slow decrease in performance as a function of R that begins at relatively small value [162]. In contrast, HebbFF has near-perfect performance for all $R < R_{\max}$, and then performance drops off quickly. However, it is likely that errors in the experiments do not reflect limitations on recognition memory but rather are due to factors such as fatigue and lack of attention that were not included in the model.

Finally, along with other recent applications of this technique [176, 175, 173, 174, 177, 178], our work demonstrates the utility of meta-learning as a tool for neuroscience discovery. We used meta-learning to optimize a network architecture and plasticity rule that solves the continual familiarity detection task, contrasted it with an alternative sub-optimal solution, and subsequently used analytic methods to understand its mechanism. A similar approach can be used for other networks, plasticity rules, datasets, and tasks.

2.4 Methods

2.4.1 HebbFF and RNN training

To set the fixed HebbFF parameters $\mathbf{W_1}$, $\mathbf{b_1}$, $\mathbf{W_2}$, $\mathbf{b_2}$, λ, η, as well as the RNN weight and bias matrices, we use the PyTorch implementation of the Adam optimizer with the suggested default hyperparameters [92]. For a single trial, we use a dataset containing T stimuli, with familiar ones appearing at a repeat interval R. We present stimuli to the network sequentially, and compute the

binary cross-entropy loss

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} y(t) \log \widehat{y}(t) + (1 - y(t)) \log (1 - \widehat{y}(t))$$

For each trial, we either use the same pre-generated length-T dataset, or we generate a new length-T dataset using the same repeat interval R. We refer to the latter case as the "infinite data" training regime since the sample space is much larger than the network would explore during training. Note that in the infinite data regime, we do not consider a validation dataset, since the training set is new every time and the training accuracy is therefore the same as the validation accuracy. In both cases, one trial corresponds to one step of gradient descent. To train the HebbFF network, the plastic matrix $\mathbf{A}(t)$ is reset to a matrix of zeros at the start of each trial. Similarly, when training the RNN, hidden unit activity is reset to zero. In practice, the plastic matrix of HebbFF reaches its steady state distribution quickly and the transient does not contribute significantly to the gradient, so any reasonable initialization can be used.

To train the HebbFF network on the augmented familiarity detection/object classification task, we simply sum the cross-entropy losses from the classifier and familiarity output units:

$$\mathcal{L} = \frac{1}{2T} \sum_{t=1}^{T} \sum_{a=1}^{2} y_a(t) \log \widehat{y}_a(t) + (1 - y_a(t)) \log (1 - \widehat{y}_a(t))$$

For every trial, we draw a new dataset from the pre-generated pool of stimuli. The class of each stimulus remains the same across datasets, but the ordering and repeats are chosen randomly each time. Although the network will have seen all of the stimuli during training in order to learn their classes, we can test generalization performance on the familiarity subtask by varying R and generating previously unseen permutations of the stimuli.

2.4.2 Bogacz-Brown model implementation

To validate it on the (non-continual) two-alternative-forced-choice (2AFC) familiarity detection task, we implement the anti-Hebbian model as described by Bogacz and Brown [161], with the

exception that the distribution of weights in the plastic weight matrix must be normalized such that its variance is equal to $\frac{1}{N}$, rather than unit variance as stated in the paper. In the encoding phase, the network is presented a sequence of P random patterns. In the testing phase, it is shown the original P patterns, as well as P novel ones. Critically, there are no plastic updates in the testing phase. A stimulus is reported as "familiar" if the output unit activity is below the mean across all $2P$ test patterns and "novel" otherwise. We see that this model performs well on the 2AFC task with a range of plasticity rates η (Figure 2.9A), so we arbitrarily choose $\eta = 0.7$ to test its performance on the continual task.

The continual task, unlike the 2AFC task, does not have an equal proportion of novel and familiar stimuli since we ensure that a stimulus is repeated at most once. So, we set the readout threshold such that an item is considered novel if it is in the f^{th} quantile of output unit activity for that trial, where f is the fraction of novel stimuli in the trial. This ensures that the fraction of stimuli reported as "novel" is equal to the true fraction of novel stimuli. In the case of equal proportions of novel and familiar stimuli, this reduces to the threshold being equal to the mean of the output unit activity for that trial.

Finally, note that unlike in the 2AFC task (Figure 2.9A), the performance of this model does not go to chance levels for large dataset sizes T in the continual task (Figure 2.3D). Rather, the true positive rate goes to zero and the false positive rate is ≈ 0.5, so accuracy is ≈ 0.33. The reason for this difference is that the second presentation of a stimulus in the continual task causes an additional plasticity event, unlike the 2AFC task where the test phase is offline. As a result, for datasets much larger than the network capacity $T \gg P^*$, the output unit activity for familiar stimuli becomes larger than the activity for novel stimuli (Figure 2.9B).

2.4.3 Training FLD and SCC decoders

To construct the Fisher linear discriminant (FLD) and spike count classifier (SCC) decoders, we first generate a dataset of length $T = 1000$. To better match the experimental dataset [165], we use multiple values of R in this single stream. For each familiar stimulus, the value of R is

drawn uniformly at random from 34 unique values, log-spaced from 1 to 100 (in practice, the results are qualitatively the same regardless of the number of items, the range, or whether the spacing is linear or logarithmic). We evaluate the trained network on this dataset and use the firing rates of the hidden layer to perform analyses analogous to those reported in [165].

We compute the readout weight and bias terms for the FLD decoder as

$$\mathbf{W_2}^{FLD} = \mathbf{\Sigma}^{-1}\left(\overline{\mathbf{h}}_{nov} - \overline{\mathbf{h}}_{fam}\right),\ b_2^{FLD} = -\mathbf{W_2}^{FLD}\cdot\frac{1}{2}\left(\overline{\mathbf{h}}_{nov} + \overline{\mathbf{h}}_{fam}\right)$$

where $\overline{\mathbf{h}}_{nov}$ and $\overline{\mathbf{h}}_{fam}$ are the average firing rates of the hidden layer for novel and familiar stimuli, respectively, and the mean covariance matrix is calculated as

$$\mathbf{\Sigma} = \frac{\mathbf{\Sigma}_{fam} + \mathbf{\Sigma}_{nov}}{2}$$

where $\mathbf{\Sigma}_{fam}$ and $\mathbf{\Sigma}_{nov}$ are the covariance matrices of the firing rates of the hidden layer for familiar and novel stimuli, respectively. The SCC decoder is a simple weighted average

$$\mathbf{W_2}^{SCC} = \frac{1}{N}\left(\overline{\mathbf{h}}_{nov} - \overline{\mathbf{h}}_{fam}\right),\ b_2^{SCC} = -\mathbf{W_2}^{SCC}\cdot\frac{1}{2}\left(\overline{\mathbf{h}}_{nov} + \overline{\mathbf{h}}_{fam}\right)$$

To get the ranking of the units for both decoders, we sort their readout weights and consider the most negative weights as the highest ranked. Note that for both decoders, the sign of the weights is flipped compared to [165], and high-ranked units have the most negative weights rather than positive. This is due to the fact that we ask the network to label familiar stimuli as $y(t) = 1$, whereas [165] readout a familiar stimulus as $y(t) = 0$. The two cases are symmetric and this does not change the results.

2.4.4 Idealized model analytic capacity derivation

For notational simplicity, we only consider the nonzero submatrices of $\mathbf{W_1}$ and $\mathbf{A}(t)$, each of which acts on its corresponding subset of the input vector $\mathbf{x}(t)$. Thus, equivalently, input layer of the idealized network is a d-dimensional vector split into two parts $\mathbf{x}(t) = [\mathbf{x_W}(t),\ \mathbf{x_A}(t)]$, of

dimension n and D respectively ($d = n + D$). Thus, the firing rate of the hidden layer is given by

$$\mathbf{h}(t) \;=\; \Theta\left(\mathbf{W_1}\mathbf{x_W}(t) + \mathbf{A}(t)\mathbf{x_A}(t) \;+\; \mathbf{b_1}\right)$$

for an $N \times n$ matrix $\mathbf{W_1}$, an $N \times D$ matrix $\mathbf{A}(t)$, and an $N \times 1$ vector $\mathbf{b_1}$. In other words, the firing rate of the i^{th} hidden unit is

$$h_i(t) = \Theta\left(\sum_{j=1}^{n} W_{ij}x_j(t) + \sum_{k=1}^{D} A_{ik}(t)x_{n+k}(t) + b\right) \tag{2.1}$$

for $i = 1, \ldots, N$, where $\Theta(\cdot)$ is the Heaviside step function, i.e. $\Theta(z) = 0$ for $z < 0$ and 1 for $z \geq 0$. We fix the value of b to be the same for all i. As before, the elements of $\mathbf{x}(t)$ are $+1$ or -1 with equal probability. We would like to specify the network parameters such that exactly one hidden neuron is active for a novel stimulus and none for familiar, which will serve as the familiarity readout mechanism.

The $N \times n$ matrix $\mathbf{W_1}$ is designed such that the vector $\mathbf{W_1}\mathbf{x_W}(t)$ has exactly one maximal entry given any such $\mathbf{x}(t)$. Importantly, this matrix must act like a hash function such that different values of $\mathbf{x_W}(t)$ result in different entries of $\mathbf{W_1}\mathbf{x_W}(t)$ attaining the maximum value. One such $\mathbf{W_1}$ is one whose rows enumerate all of the binary length-n strings consisting of entries $+1$ and -1. This sets the number of rows N to be equal to the total number of such strings, $N = 2^n$. To set the overall scale of the input current (the term inside the nonlinearity), we scale this matrix by a factor K, to be determined later. For example, if $n = 3$,

$$\mathbf{W_1} = K \begin{bmatrix} +1 & +1 & +1 \\ +1 & +1 & -1 \\ +1 & -1 & +1 \\ +1 & -1 & -1 \\ -1 & +1 & +1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \\ -1 & -1 & -1 \end{bmatrix}$$

Thus, we have $\sum_{j=1}^{n} W_{ij}x_j(t) = Kn$ for exactly one value of i, specifically the row where

$W_{ij} = x_j(t)$ for all j. This is the unique maximal value and will correspond to a different row for each instance of $\mathbf{x_W}(t)$. Subsequently, $\sum_{j=1}^{n} W_{ij}x_j(t) = K(n-2)$ for n values of i, specifically those where $W_{ij} \neq x_j(t)$ for exactly one j, and so on. This structure explains the multi-modal distribution of $\mathbf{W_1 x_W}(t) + \mathbf{b_1}$ in Figure 2.5D and by extension that of $\mathbf{W_1 x}(t) + \mathbf{b_1}$ in Figure 2.3G-I.

Assuming that the vector $\mathbf{A}(t)\mathbf{x_A}(t)$ is zero-mean with sufficiently small variance (this will be made rigorous shortly), we can now choose the scalar offset b such that exactly one element of $\mathbf{h}(t)$ is equal to 1 and all others are zero.

The $N \times D$ matrix $\mathbf{A}(t)$ is updated at every timestep by $\mathbf{A}(t+1) = \lambda\mathbf{A}(t) - \eta\mathbf{h}(t)\mathbf{x_A}(t)^T$, where the plasticity rate η is now restricted to be positive, corresponding to an anti-Hebbian learning rule. Considering one entry in this matrix and unrolling this recurrence, we find that

$$\begin{aligned} A_{ik}(t+1) &= \lambda A_{ik}(t) - \eta h_i(t) x_{n+k}(t) \\ &= \lambda^{t+1} A_{ik}(0) - \eta \sum_{t'=0}^{t} \lambda^{t-t'} h_i\left(t'\right) x_{n+k}\left(t'\right) \\ &\approx -\eta \sum_{t'=0}^{t} \lambda^{t-t'} h_i\left(t'\right) x_{n+k}\left(t'\right) \end{aligned}$$

We can now consider the middle term of Eq. 2.1, which we denote by $\varepsilon_i(t)$. We consider it as a random variable and compute its mean and variance. By definition, we have

$$\begin{aligned} \varepsilon_i(t) &= \sum_{k=1}^{D} A_{ik}(t) x_{n+k}(t) \\ &= \sum_{k=1}^{D} \left(-\eta \sum_{t'=0}^{t-1} \lambda^{t-1-t'} h_i\left(t'\right) x_{n+k}\left(t'\right) \right) x_{n+k}(t) \\ &= -\eta \sum_{t'=0}^{t-1} \lambda^{t-1-t'} h_i\left(t'\right) \sum_{k=1}^{D} x_{n+k}\left(t'\right) x_{n+k}(t) \end{aligned} \tag{2.2}$$

In the case where $\mathbf{x}(t)$ is novel, $x_{n+k}\left(t'\right)$ and $x_{n+k}(t)$ are independent Bernoulli random variables

that take on values ± 1 with probability 0.5. Thus, $X_k\left(t'\right) = x_{n+k}\left(t'\right) x_{n+k}(t)$ is also a Bernoulli random variable with the same distribution, zero mean and unit variance, so

$$\varepsilon_i(t) = -\,\eta \sum_{t'=0}^{t-1} \lambda^{t-1-t'} h_i\left(t'\right) \sum_{k=1}^{D} X_k(t')$$

Since the entries of $\mathbf{x}(t)$ are independent by definition, the $X_k(t')$ are also independent across *k*, so summing over these indices, the variances add. Therefore, $X\left(t'\right) = \sum_{k=1}^{D} X_k\left(t'\right)$ is a random variable with mean 0 and variance D, and

$$\varepsilon_i(t) = -\,\eta \sum_{t'=0}^{t-1} \lambda^{t-1-t'} h_i\left(t'\right) X(t')$$

Next, we need the statistics of the term $h_i\left(t'\right)$. Since it is a function of the random variable $\mathbf{x}(t)$, we also consider it as a random variable. Let f_{eff} denote the fraction of stimuli reported as "novel" by the network. Note that there are two ways for a network to report a stimulus as "novel" – by correctly identifying a novel stimulus ("true negative"), or incorrectly identifying a familiar one ("false negative") – so if we let f denote the true fraction of novel input stimuli, we have

$$f_{\text{eff}} \;=\; P_{TN} f \;+\; P_{FN}(1-f) = (1-P_{FP}) f \;+\; (1-P_{TP})(1-f)$$

where $P_{TN},\; P_{FN}, P_{TP}$ and P_{FP} are the true negative, false negative, true positive, and false positive rates, respectively. Since by design there is exactly one hidden unit active for a novel stimulus, we have $h_i\left(t'\right) = 1$ with probability $\frac{f_{\text{eff}}}{N}$, and $h_i\left(t'\right) = 0$ with probability $1 - \frac{f_{\text{eff}}}{N}$. So, $h_i\left(t'\right)$ is a Bernoulli random variable with mean $\frac{f_{\text{eff}}}{N}$ and variance $\frac{f_{\text{eff}}}{N}\left(1 - \frac{f_{\text{eff}}}{N}\right)$. Now, we let $H_i\left(t'\right) = h_i\left(t'\right) X\left(t'\right)$, so

$$\varepsilon_i(t) = -\,\eta \sum_{t'=0}^{t-1} \lambda^{t-1-t'} H_i\left(t'\right)$$

Although $h_i\left(t'\right)$ is, in principle, a function of $\mathbf{x}(t')$, we assume they are independent. Since $X\left(t'\right)$ is zero-mean, the mean of $H_i\left(t'\right)$ is also zero. Using the identity $\text{var}(XY) = \text{var}(X)\text{var}(Y) +$

$\text{var}(X)\mathbb{E}^2[Y] + \text{var}(Y)\mathbb{E}^2[X]$, which holds for independent random variables X and Y, we have that the variance of $H_i\left(t'\right)$ is $\frac{f_{\text{eff}}D}{N}$. Finally, for convenience we can rewrite this as

$$\varepsilon_i(t) = -\eta\sqrt{\frac{f_{\text{eff}}D}{N}}\sum_{t'=0}^{t-1}\lambda^{t-1-t'}\xi_i\left(t'\right)$$

where $\xi_i\left(t'\right)$ is a zero-mean, unit-variance random variable. Furthermore, we now see that by the Central Limit Theorem $\varepsilon_i(t)$ is a Gaussian random variable since we are considering the steady-state performance at large t, so we can take $t \to \infty$.

We can now compute the mean and variance of $\varepsilon_i(t)$. First, since $x_{n+k}(t)$ is zero-mean and independent of $A_{ik}(t)$,

$$\mathbb{E}[\varepsilon_i(t)] = \mathbb{E}\left[\sum_{k=1}^{D}A_{ik}(t)x_{n+k}(t)\right] = 0$$

To compute the variance,

$$\begin{aligned}
\text{var}\left(\varepsilon_i(t)\right) = \ & \mathbb{E}[\varepsilon_i^2(t)] - \mathbb{E}^2[\varepsilon_i(t)] = \mathbb{E}[\varepsilon_i^2(t)] \\
&= \mathbb{E}\left[\left(-\eta\sqrt{\frac{f_{\text{eff}}D}{N}}\sum_{t'=0}^{t-1}\lambda^{t-1-t'}\xi_i\left(t'\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(-\eta\sqrt{\frac{f_{\text{eff}}D}{N}}\sum_{t'=0}^{t-1}\lambda^{t-1-t'}\xi_i\left(t'\right)\right)\left(-\eta\sqrt{\frac{f_{\text{eff}}D}{N}}\sum_{t''=0}^{t-1}\lambda^{t-1-t''}\xi_i\left(t''\right)\right)\right] \\
&= \eta^2\frac{f_{\text{eff}}D}{N}\sum_{t'=0}^{t-1}\sum_{t''=0}^{t-1}\lambda^{t-1-t'}\lambda^{t-1-t''}\mathbb{E}\left[\xi_i\left(t'\right)\xi_i\left(t''\right)\right]
\end{aligned}$$

In general, we have $\mathbb{E}\left[\xi_i\left(t'\right)\xi_i\left(t''\right)\right] = 1$ for $t' = t''$ since $\xi_i\left(t'\right)$ is a zero-mean, unit-variance random variable. For $t' \neq t''$, we again make a simplifying independence assumption. In principle, $\xi_i\left(t''\right)$ is not independent of $\xi_i\left(t'\right)$ since $h_i\left(t''\right)$ depends on $h_i\left(t'\right)$ for $t'' > t'$ through the memory stored in the $\mathbf{A}(t)$ matrix. This dependence, however, is sufficiently weak, so we let $\mathbb{E}\left[\xi_i\left(t'\right)\xi_i\left(t''\right)\right] = 0$ for $t' \neq t''$. As a result, the double-sum collapses and we have

$$\text{var}\left(\varepsilon_i(t)\right) = \eta^2\frac{f_{\text{eff}}D}{N}\sum_{t'=0}^{t-1}\lambda^{2(t-1-t')} = \eta^2\frac{f_{\text{eff}}D}{N}\frac{1-\lambda^{2t}}{1-\lambda^2}$$

where the second equality comes from the standard geometric series. As before, since we are considering the steady-state with $t \to \infty$, we have $\gamma^{2t} \to 0$, so

$$\text{var}\left(\varepsilon_i(t)\right) = \eta^2 \frac{f_{\text{eff}} D}{N} \frac{1}{1-\lambda^2}$$

Thus, for a novel input $\mathbf{x}(t)$ we can write

$$\varepsilon_i(t) = \xi_i \eta \sqrt{\frac{f_{\text{eff}} D}{N\left(1-\lambda^2\right)}} \tag{2.3}$$

for all i, where ξ_i is a zero-mean, unit-variance Gaussian random variable, since $\varepsilon_i(t)$ is Gaussian.

For a familiar stimulus, where $\mathbf{x}(t) = \mathbf{x}(t-R)$, clearly $x_{n+k}\left(t'\right)$ and $x_{n+k}(t)$ are no longer independent for $t' = t - R$. Thus, we consider this term separately, rewriting the sum in Eq. 2.2 as

$$\varepsilon_i(t) = -\eta \lambda^{t-1-(t-R)} h_i(t-R) \sum_{k=1}^{D} x_{n+k}(t-R) x_{n+k}(t) - \eta \sum_{\substack{t'=0 \\ t' \neq t-R}}^{t-1} \lambda^{t-1-t'} h_i\left(t'\right) \sum_{k=1}^{D} x_{n+k}\left(t'\right) x_{n+k}(t)$$

Assuming no errors, by design, $h_i(t-R) = 1$ for exactly one neuron i, since the stimulus at time $t - R$ was guaranteed to be novel (we enforce that a stimulus is repeated at most once in this task). We consider the statistics of $\varepsilon_i(t)$ for this particular neuron. In the first term, the sum $\sum_{k=1}^{D} x_{n+k}(t-R) x_{n+k}(t) = D$ since by assumption $x_{n+k}(t) = x_{n+k}(t-R)$ for all k. The second term has the same distribution as the one for a novel input since we have only removed one term from the sum and t is large. Thus, for a familiar stimulus we can write

$$\varepsilon_i(t) = -\eta \lambda^{R-1} D + \xi_i \eta \sqrt{\frac{f_{\text{eff}} D}{N\left(1-\lambda^2\right)}}$$

for exactly one value of i, where ξ_i is a zero-mean, unit-variance random variable as before. For all other values of i, Eq. 2.3 holds.

Having established the statistics of the hidden layer input currents for a novel and a familiar stimulus, we can now write down the conditions for the model to work, use them to find the optimal

values of the parameters and calculate the true positive and false positive probabilities, and compute the capacity – the largest value of R for which the error is below a predetermined threshold. First, to ensure that exactly one unit is active for a novel stimulus (true negative), since we are using a step function nonlinearity, we must have the largest input current take on a positive value (since ξ_i is an identically distributed standard normal random variable for every neuron, for simplicity we suppress the index i),

$$Kn + \xi\eta\sqrt{\frac{f_{\text{eff}}D}{N(1-\lambda^2)}} + b > 0$$

and second-largest to be below zero,

$$K(n-2) + \xi\eta\sqrt{\frac{f_{\text{eff}}D}{N(1-\lambda^2)}} + b < 0$$

Second, to ensure there are no units active for a familiar stimulus (true positive),

$$Kn - \eta\lambda^{R-1}D + \xi\eta\sqrt{\frac{f_{\text{eff}}D}{N(1-\lambda^2)}} + b < 0$$

For sufficiently large R, i.e. if $\eta\lambda^{R-1}D < 2K$, the third of these conditions implies the second. Since we are interested in maximizing R, we only consider the first and third conditions. Furthermore, note that these conditions are overparameterized. If we divide all three equations by η (e.g. let $k = \frac{K}{\eta},\ B = \frac{b}{\eta}$), we can eliminate this free parameter. In other words, for any value of η we can scale K and b proportionally to satisfy the conditions, so for simplicity we choose $\eta = 1$. Similarly, the term $Kn + b$ can be replaced by a single parameter since for any choice of K we can rescale b to keep this sum constant. To ensure that the condition $\eta\lambda^{R-1}D < 2K$ holds for all R, we can choose $K = D$. For convenience, we also let $b = \beta D$ and $\sqrt{\frac{f_{\text{eff}}D}{N(1-\lambda^2)}} = \alpha_\lambda D$, the subscript indicating explicit dependence on λ. Dividing both inequalities by D, the conditions simplify to

$$n + \alpha_\lambda\xi + \beta > 0,\ \ n + \beta + \alpha_\lambda\xi - \lambda^{R-1} < 0$$

The accuracy, i.e. probability of a correct response, is given by $P_{correct} = (1-f)P_{TP} + fP_{TN}$.

For convenience, we compute the false positive instead of the true negative rate, noting that $P_{TN} = 1 - P_{FP}$. The false positive and true positive rates are given by

$$P_{FP} = \mathbb{P}\left[\xi < -\frac{n+\beta}{\alpha_\lambda}\right], \;\; P_{TP} = \mathbb{P}\left[\xi < -\frac{n+\beta-\lambda^{R-1}}{\alpha_\lambda}\right]$$

Since ξ is a standard Normal random variable, $\mathbb{P}[\xi < z] = \frac{1}{2}\text{erfc}\left(-\frac{z}{\sqrt{2}}\right)$, so

$$P_{FP} = \frac{1}{2}\text{erfc}\left(\frac{n+\beta}{\alpha_\lambda\sqrt{2}}\right), \;\; P_{TP} = \frac{1}{2}\text{erfc}\left(\frac{n+\beta-\lambda^{R-1}}{\alpha_\lambda\sqrt{2}}\right)$$

We would now like to set the optimal values of λ and β which maximize R, given a desired true positive and false positive probability $P^*_{FP},\ P^*_{TP}$. Note that fixing these probabilities also fixes $f_{\text{eff}} = f^* = \ (1 - P^*_{FP})f \ + \ (1 - P^*_{TP})(1 - f)$. Rearranging the previous equations, we get

$$\frac{n+\beta}{\alpha_\lambda} = \ \sqrt{2}\text{erfc}^{-1}\left(2P^*_{FP}\right), \;\; \frac{n+\beta-\lambda^{R-1}}{\alpha_\lambda} = \ \sqrt{2}\text{erfc}^{-1}(2P^*_{TP})$$

The first equality sets the value for β. To determine λ, we substitute β into the second equality to get

$$\sqrt{2}\text{erfc}^{-1}\left(2P^*_{FP}\right) - \frac{\lambda^{R-1}}{\alpha_\lambda} = \sqrt{2}\text{erfc}^{-1}(2P^*_{TP})$$

For notational convenience, let $E = \sqrt{2}\left[\text{erfc}^{-1}\left(2P^*_{FP}\right) - \text{erfc}^{-1}(2P^*_{TP})\right]$. Using the definition of α_λ and $f_{\text{eff}} = f^*$, we have $\lambda = \sqrt{1 - \frac{f^*}{\alpha_\lambda^2 ND}}$. Rearranging, we have

$$\left(1 - \frac{f^*}{\alpha_\lambda^2 ND}\right)^{\frac{R-1}{2}} \ = \alpha_\lambda E$$

Assuming N and D are large (so λ is close to 1), we can use the first-order Taylor expansion $\exp(-z) \approx 1 - z$ for the term in parentheses (this will be necessary to get a closed-form expression for the optimal λ) and solve for R

$$\exp\left(-\frac{f^*}{\alpha_\lambda^2 ND} \cdot \frac{R-1}{2}\right) \ = \alpha_\lambda E \ \Rightarrow \ R = 1 + \frac{2D\alpha_\lambda^2}{f^*}\ln\left(\frac{1}{\alpha_\lambda E}\right)$$

Setting $\frac{dR}{d\lambda} = 0$ and solving for λ gives the optimum

$$\lambda = \sqrt{1 - \frac{eE^2 f^*}{ND}}$$

$$R_{\max} = 1 + \frac{ND}{eE^2 f^*}$$

where f^* and E are constants that depend on P^*_{FP} and P^*_{TP} (f^* also depends on the true fraction of novel stimuli f). For instance, if we impose that $P^*_{FP} = 0.01$ and $P^*_{TP} = 0.99$, with our value of $f = \frac{2}{3}$, we get

$$\begin{aligned}
R_{\max} &= 1 + \frac{ND}{e{\cdot}2 \cdot \left[\text{erfc}^{-1}\left(2P^*_{FP}\right) - \text{erfc}^{-1}\left(2P^*_{TP}\right)\right]^2 \cdot \left[(1 - P^*_{FP})f \ + \ (1 - P^*_{TP})(1 - f)\right]} \\
&= 1 + \frac{ND}{2e\left[1.645 - (-1.645)\right]^2 \left[0.98f \ + \ 0.01\right]} \\
&= 1 + \frac{0.017ND}{0.98f \ + \ 0.01} \\
&= 1 + 0.026ND
\end{aligned}$$

It is clear that that the capacity scales in proportion to the number of plastic synapses in the network. Furthermore, since $d = n + D$, i.e. $D = d - \log_2(N)$, the capacity scales in proportion to the total number of synapses d, as long as $D \gg n$.

$$R_{\max} = O(ND) = O\left(N(d - n)\right) = O(Nd - N\log N)) = O(Nd)$$

Finally, note that the equations for P_{FP} and P_{TP} are a function of f_{eff} due to the α_λ parameter, and therefore recursively depend on P_{FP} and P_{TP}. We cannot compute the closed-form solution for these, but we can approximate the values with arbitrary accuracy by iterating through this recurrence until convergence to the fixed point. As the initial value for the recurrence, we use P_{FP} and P_{TP} computed using $f_{\text{eff}} = f$, i.e. assuming no errors.

2.5 Supplementary figures

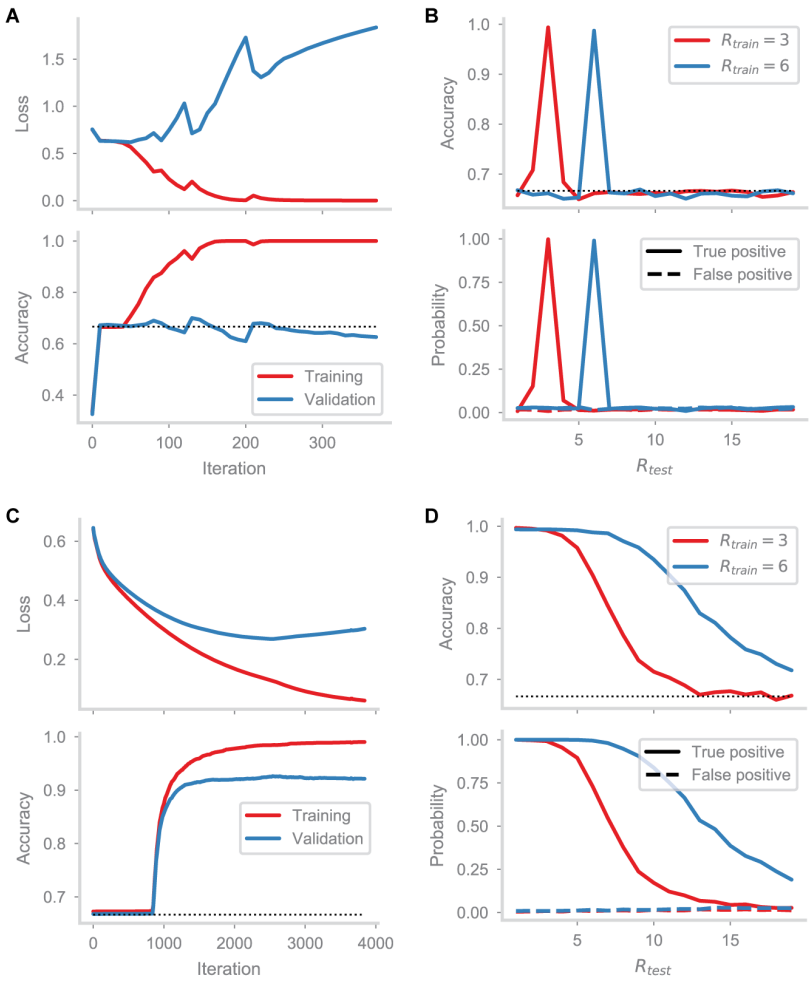


Figure 2.8: HebbFF and RNN comparison, matching total number of dynamic variables. (A,B) RNN performance as in Figure 2.2A-B, with $d = 25$ and $N = 625$. (C,D) HebbFF performance as Figure 2.3A-B, with $d = 25$ and $N = 25$. The number of plastic synapses in the HebbFF network $N * d = 625$ is the same as the number of recurrent units in the RNN, matching the number of dynamic variables between the networks rather than the number of neurons. HebbFF still shows better generalization in both training scenarios.

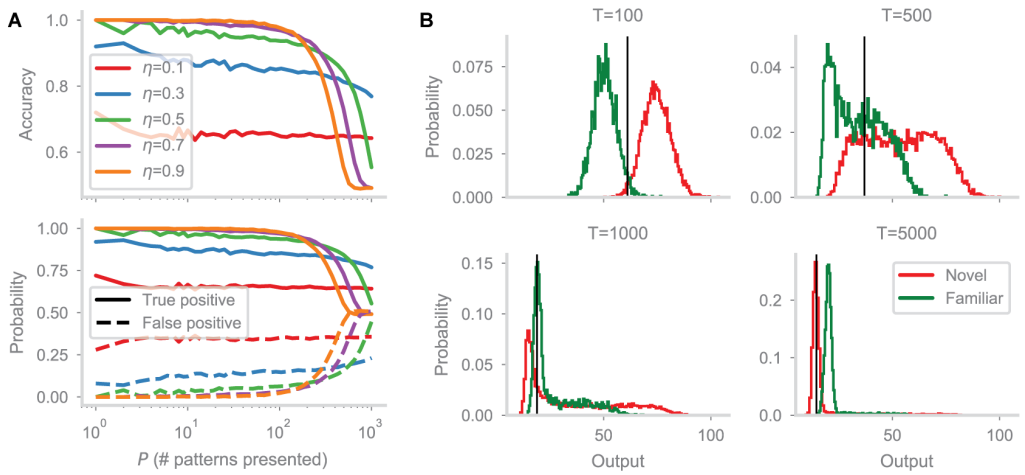


Figure 2.9: Validation of network from [161]. Related to Figure 2.3, section 2.4. (A) Performance of the anti-Hebbian network from (Bogacz and Brown, 2003) ($d = N = 100$) on the non-continual familiarity detection task. The network is shown P uncorrelated randomly generated patterns, and tested on all the presented patterns as well as P novel ones. If the network's readout falls below a threshold, the corresponding input is classified as "familiar," otherwise "novel." The threshold is set such that that exactly half of the test inputs are classified as "familiar." The plot shows the network's performance for a range of learning rates η as a function of the number of patterns presented. (B) Performance of the same network ($\eta = 0.7$) on the continual task. Plots show the probability distribution of the network's readout for novel (red) and familiar (green) stimuli. Threshold for familiarity (vertical black line) is set such that the top f of outputs are classified as novel (fraction f equal to proportion of novel stimuli in dataset), analogous to that in [161]. Network performance declines as the distributions get closer together for increasing dataset size T. For very large dataset sizes, e.g. $T = 5000$, the readout for familiar stimuli is actually higher than that for novel, which causes overall performance (Figure 2.3D) to fall below chance.

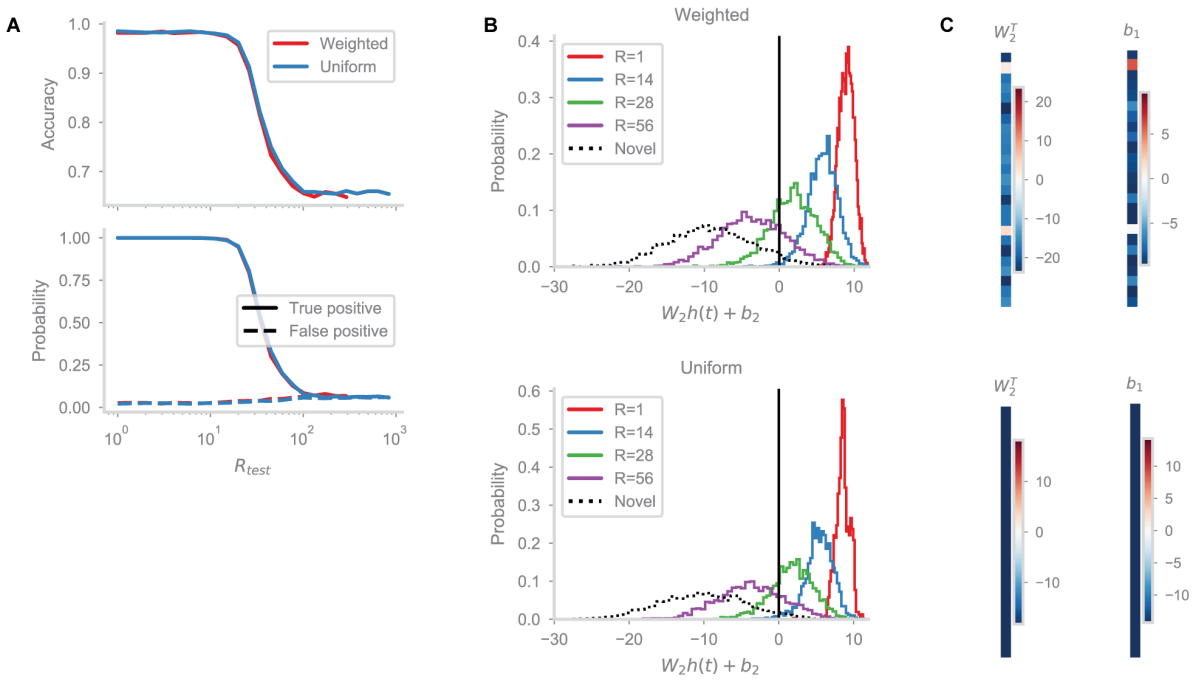


Figure 2.10: Comparison of uniform and trained readout. Related to Figure 2.3. (A) Generalization performance of a trained HebbFF network ($d = N = 25$) with a fully trainable readout matrix, i.e. a weighted average of the hidden layer activity (red) and a version where all the entries of the readout matrix are constrained to be equal, i.e. a scaled average of the hidden layer (blue). Performance is not affected by this choice of output weights. (B) Distributions of the output unit activity (prior to applying the nonlinearity) for a trained network, for several test values of R. The output distributions are almost identical for both versions of the readout. (C) Examples of the $\mathbf{W_2}$ readout matrix, as well as the bias $\mathbf{b_1}$ for the weighted (top) and uniform (bottom) readouts. Values for both are negative, indicating a qualitatively similar readout mechanism. Note that the weight matrices are plotted transposed for visualization. Also note that anti-Hebbian plasticity is the preferred form of plasticity for both versions of the readout.

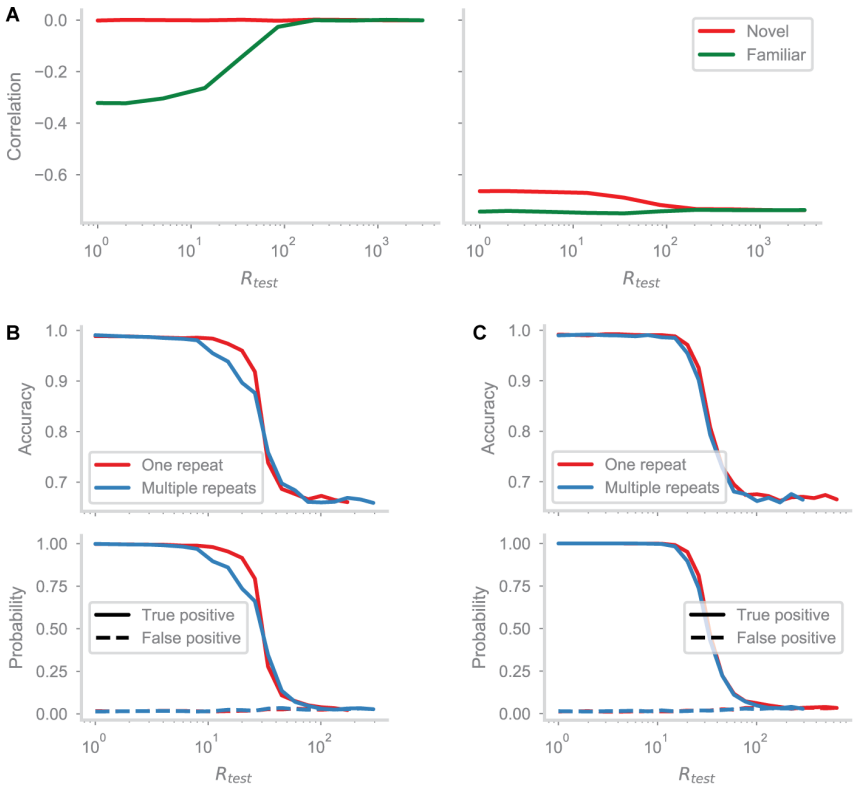


Figure 2.11: Idealized and HebbFF model differences. Related to Figure 2.5. (A) Correlation between the hidden layer input currents due to static and plastic synapses for the idealized model (left) and HebbFF (right) for novel (red) and familiar (green) stimuli as a function of delay interval ($d = 25,\ N = 32$). In the idealized model, due to the split static and plastic synapses, the currents are uncorrelated for novel stimuli and familiar stimuli at long delay intervals. At short intervals, the two input currents are anti-correlated, which enables repetition suppression. In the HebbFF model, there is anti-correlation in both cases, although there is still less anti-correlation for novel stimuli. (B) Performance of the idealized network on the continual familiarity detection task with familiar stimuli repeated either exactly once, as used throughout this work (red), or multiple times (blue). Since there is exactly zero hidden unit activity for a familiar stimulus it does not get reinforced in memory, and less likely to be recognized on its second and subsequent repetitions. (C) The HebbFF network, trained to maximum capacity on the single-repeat task does not suffer any loss in performance due to multiple repeats.

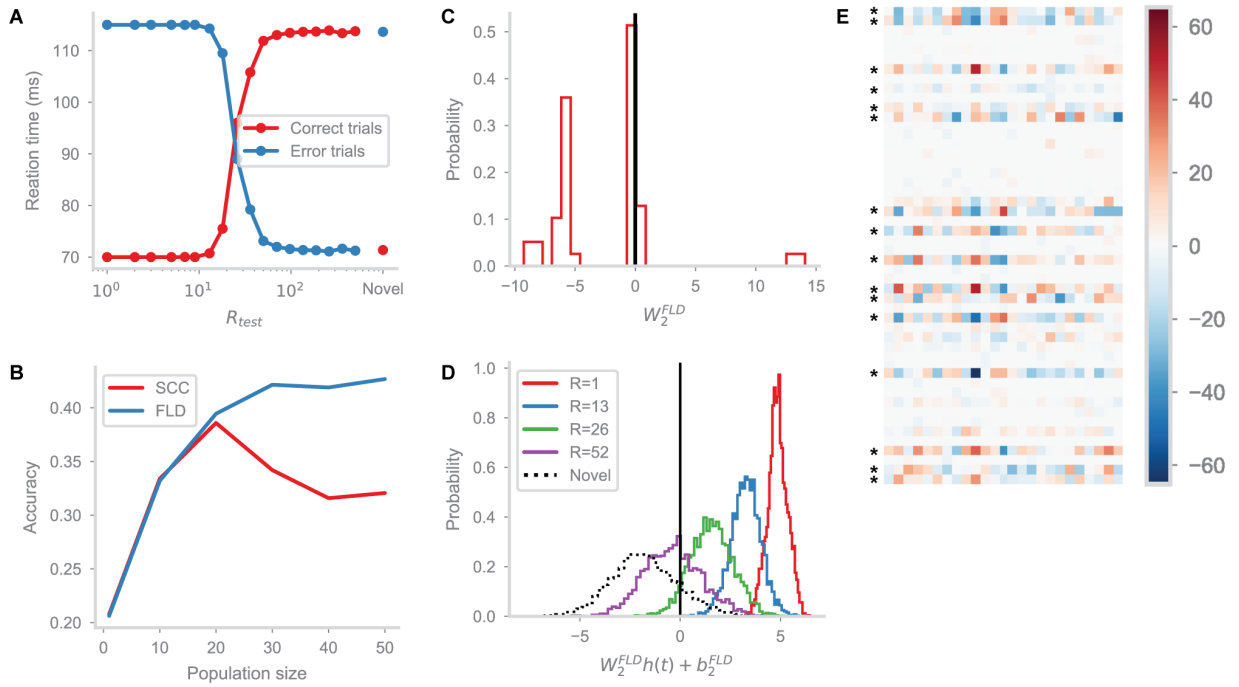


Figure 2.12: Behavior on augmented task. (A-D) Correspond to the right-hand side plots of Figure 2.6A-D, but for the classifier-augmented HebbFF network ($d = 25,\ N = 50$) performing binary classification and familiarity detection simultaneously. (E) The weight matrix $\mathbf{W_1}$ of the HebbFF network trained on the augmented task, requiring simultaneous classification and familiarity detection. Hidden units split into "classification" and "familiarity" units, with classification units (marked with asterisks) having very strong input weights to overcome the noise from the plastic $\mathbf{A}(t)$ matrix.

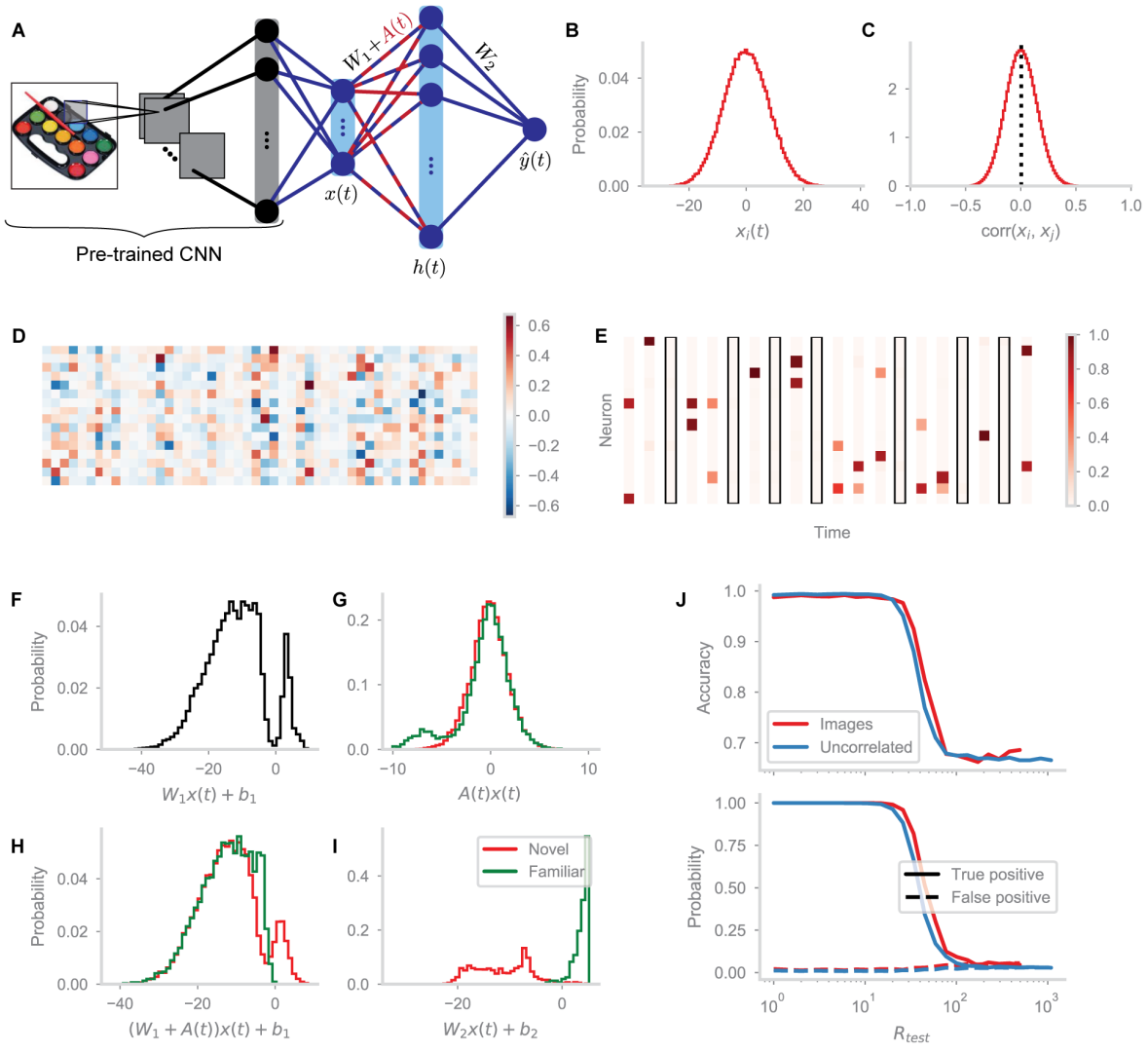


Figure 2.13: Performance on real-world images. Subplots as in Figure 2.7, but using a trained fully-connected linear layer to transform the activations of the CNN's penultimate layer into the inputs of HebbFF ($R_{train} = R_{test} = 19$).

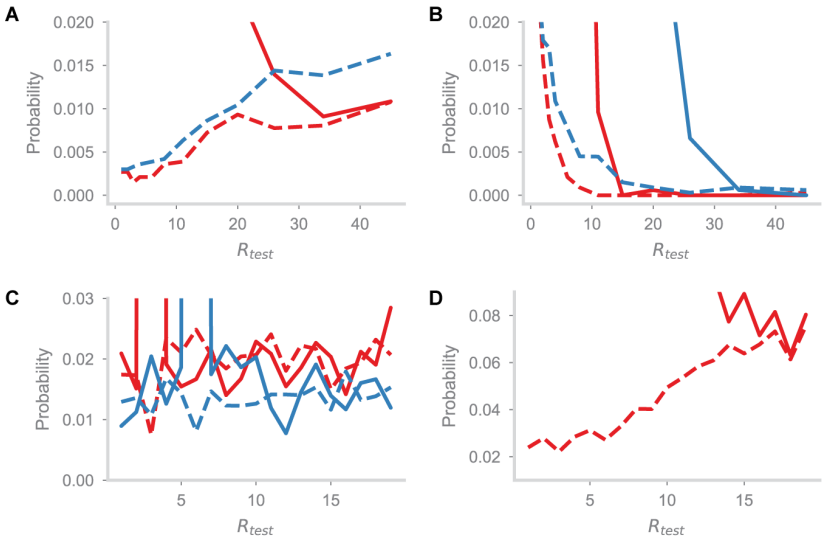


Figure 2.14: Close-up of false positive probability curves. (A) Same as Figure 2.3C, bottom plot, zoomed. HebbFF with anti-Hebbian plasticity shows an increase in false positive probability (dashed curves) with increasing R, and is a sufficient mechanism to explain this effect. This occurs because false negatives due to spuriously elevated activity in the hidden layer cause storage of a familiar stimulus in additional neurons, as determined by the addressing function. Subsequently, a novel stimulus is more likely to generate a false positive response due to a collision in the addressing function. (Note that although this effect increases false positives, it may also boost the true positive rate since the memory gets more robustly encoded). Prior experimental paradigms have not measured this effect because each trial had familiar stimuli interleaved at various delay intervals, so novel stimuli could not be separated scored depending on the difficulty of the dataset and false positive probability was reported in aggregate. (B) Same as Figure 2.3B. Hebbian plasticity does not exhibit an increase in false positives. (C) Same as Figure 2.2A. RNNs trained with a single R also do not. (D) Same as Figure 2.2C. An RNN trained on multiple R's also shows this effect. An increase in false probabilities may be caused by an alternative mechanism. Thus, anti-Hebbian plasticity is sufficient but not necessary for this effect.

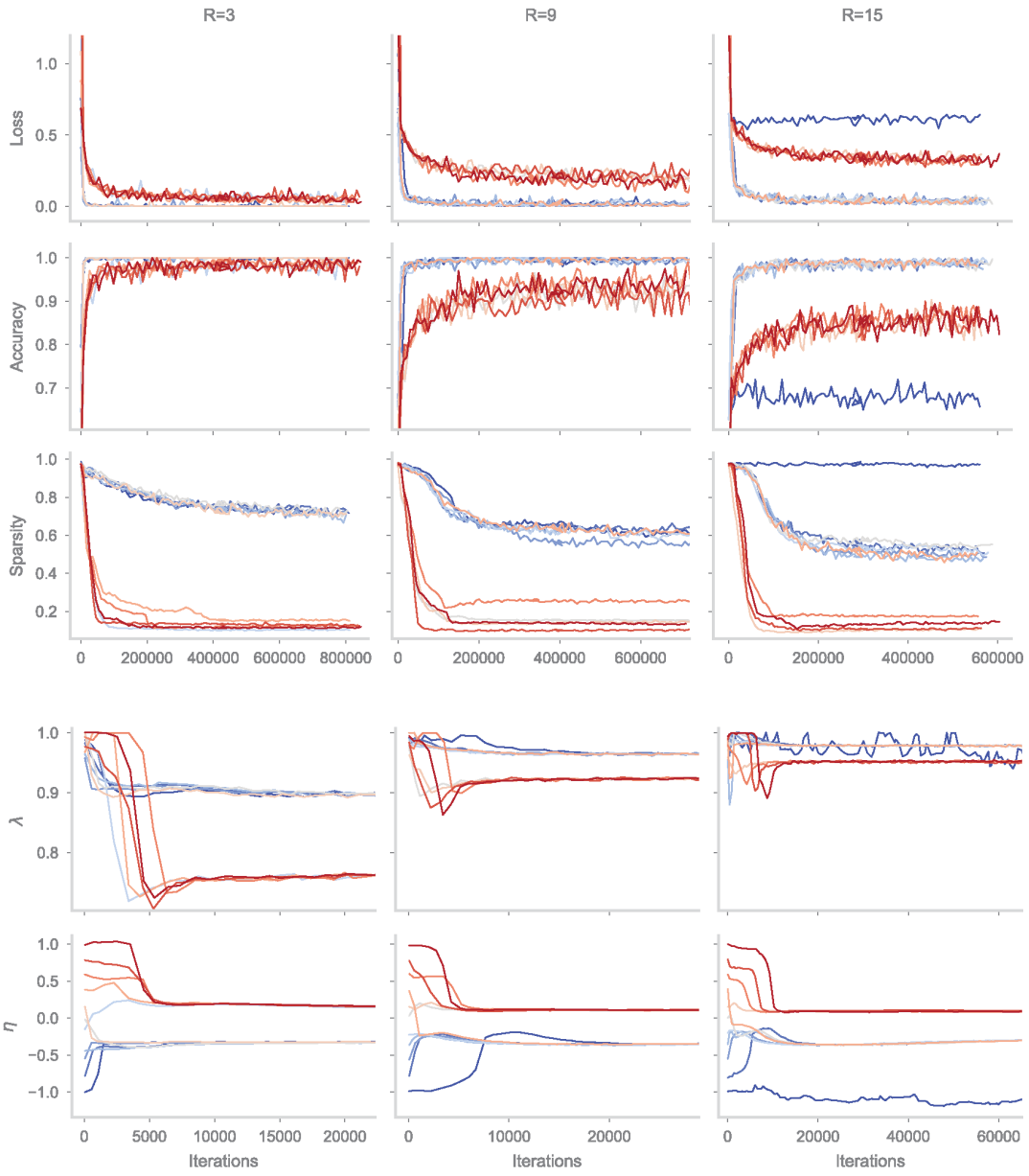


Figure 2.15: Evolution of parameters. Related to Figure 2.3. HebbFF is initialized with $\lambda = 0.9$, η between -1 (blue) and $+1$ (red), $\mathbf{W_1}$ drawn from a zero-mean normal distribution with variance N, and trained on $R = 3$, (left) 9, (middle), or 15 (right). Loss, accuracy, sparsity of $\mathbf{W_1}$ (fraction of entries below a threshold 0.01), λ and η are plotted over the course of training for each condition (λ and η plotted for fewer iterations to better visualize their trajectories, since they converge sooner). For each value of R there exist only two qualitatively different solutions – Hebbian and anti-Hebbian. All networks with sufficiently small (not necessarily negative) η converge to an anti-Hebbian solution. Networks with a Hebbian plasticity rule converge more slowly and to a lower accuracy/higher loss than those with an anti-Hebbian rule, although training may diverge for larger R (right, dark blue curve). Anti-Hebbian networks and networks trained with larger R have λ closer to 1 and sparser $\mathbf{W_1}$ matrices.

Chapter 3: Biological learning in key-value memory networks

The work presented in this chapter was done in collaboration with Ching Fang (equal contribution), Annapurna Vadaparty, and Guangyu Robert Yang; published in [21]. We are particularly grateful for the mentorship of Larry Abbott. We also thank Stefano Fusi, James Whittington, Emily Mackevicius, and Dmitriy Aronov for helpful discussions. Thanks to David Clark for bringing Bidirectional Associative Memory to our attention.

In neuroscience, classical Hopfield networks are the standard biologically plausible model of long-term memory, relying on Hebbian plasticity for storage and attractor dynamics for recall. In contrast, memory-augmented neural networks in machine learning commonly use a key-value mechanism to store and read out memories in a single step. Such augmented networks achieve impressive feats of memory compared to traditional variants, yet their biological relevance is unclear. We propose an implementation of basic key-value memory that stores inputs using a combination of biologically plausible three-factor plasticity rules. The same rules are recovered when network parameters are meta-learned. Our network performs on par with classical Hopfield networks on autoassociative memory tasks and can be naturally extended to continual recall, heteroassociative memory, and sequence learning. Our results suggest a compelling alternative to the classical Hopfield network as a model of biological long-term memory.

3.1 Introduction

Long-term memory is an essential aspect of our everyday lives. It is the ability to rapidly memorize an experience or item, and to retain that memory in a retrievable form over a prolonged duration (days to years in humans). Neural networks capable of long-term memory have been

studied in both neuroscience and machine learning, yet a wide gap remains between the mechanisms and interpretations of the two traditions.

In neuroscience, long-term associative memory is typically modeled by variants of Hopfield networks [187, 188, 189]. Rooted in statistical physics, they are one of the earliest and best known class of neural network models. A classical Hopfield network stores an activation pattern $\boldsymbol{\xi}$ by strengthening the recurrent connections $\boldsymbol{W}$ between co-active neurons using a biologically-plausible Hebbian plasticity rule,

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{\xi}\boldsymbol{\xi}^{\mathsf{T}} \tag{3.1}$$

and allows retrieval of a memory from a corrupted version through recurrent attractor dynamics,

$$\boldsymbol{x}_{t+1} = \operatorname{sign}(\boldsymbol{W}\boldsymbol{x}_t) \tag{3.2}$$

thereby providing a content-addressable and pattern-completing autoassociative memory.

In a more recent parallel thread in machine learning, various memory networks have been devised to augment traditional neural networks [126, 190, 191, 192, 193]. Memory-augmented neural networks utilize a more stable external memory system analogous to computer memory, in contrast to more volatile storage mechanisms such as recurrent neural networks [194]. Many memory networks from this tradition can be viewed as consisting of memory slots where each slot can be addressed with a *key* and returns a memory *value*, although this storage scheme commonly lacks a mechanistic interpretation in terms of biological processes.

Key-value networks date back to at least the 1980s with Sparse Distributed Memory (SDM) as a model of human long-term memory [73, 60]. Inspired by random-access memory in computers, it is at the core of many memory networks recently developed in machine learning [126, 195, 190, 196, 192]. A basic key-value network contains a key matrix $\boldsymbol{K}$ and a value matrix $\boldsymbol{V}$. Given a query

vector $\widetilde{\boldsymbol{x}}$, a memory read operation will retrieve an output $\boldsymbol{y}$ as

$$\begin{aligned} \boldsymbol{h} &= f(\boldsymbol{K}\widetilde{\boldsymbol{x}}) \\ \boldsymbol{y} &= \boldsymbol{V}\boldsymbol{h} \end{aligned} \tag{3.3}$$

where f is an activation function that sparsifies the hidden response $\boldsymbol{h}$.

Variations exist in the reading mechanisms of key-value memory networks. For example, f may be the softmax function [190], making memory retrieval equivalent to the "key-value attention" [197] used in recent natural language processing models [198]. It has also been set as the step function [73] and hard-max function [199]. There is an even greater variation across writing mechanisms of memory networks. Some works rely on highly flexible mechanisms where an external controller learns which slots to write and overwrite [195, 126], although appropriate memory write strategies can be difficult to learn. Other works have used simpler mechanisms where new memories can be appended sequentially to the existing set of memories [190, 200, 201] or written through gradient descent [193, 191, 192, 202]. [60] updates the value matrix through Hebbian plasticity, but fixes the key matrix. [191] turns memory write into a key-target value regression problem, and updates an arbitrary feedforward memory network using metalearned local regression targets.

A clear advantage of key-value memory over classical Hopfield networks is the decoupling of memory capacity from input dimension [60, 203]. In classical Hopfield networks, the input dimension determines the number of recurrent connections, and thus upper bounds the capacity. In key-value memory networks, by increasing the size of the hidden layer, the capacity can be much larger when measured against the input dimension (although the capacity per connection is similar).

There exists a gap between these two lines of research on neural networks for long-term memory – classical Hopfield networks in the tradition of computational neuroscience and key-value memory in machine learning. In our work, we study whether key-value memory networks used in machine learning can provide an alternative to classical Hopfield networks as biological models for long-term memory.

Modern Hopfield Networks (MHN) [202, 203, 201, 204] begin to address this issue, suggesting a

neural network architecture for readout of key-value pairs from a synaptic weight matrix, but lacking a biological learning mechanism. MHNs implement an autoassociative memory (key is equal to the value, so $\boldsymbol{V} = \boldsymbol{K}^\intercal$) [201, 203], but they can be used for heteroassociative recall by concatenating the key/value vectors and storing the concatenated versions instead. To query the network, keys can be clamped, and only the hidden and value neurons updated [202]. With a particular choice of activation function and one-step dynamics, this architecture is mathematically equivalent to a fully connected feedforward network [202], and thus analogous to the memory architecture proposed by [60].

It remains unclear whether a biological mechanism can implement the memory write. We will show that such biologically-plausible plasticity rules exist. We consider a special case of the MHN architecture and introduce a biologically plausible learning rule, using local three-factor synaptic plasticity [205], as well as respecting the topological constraints of biological neurons with spatially separated dendrites and axons. We first suggest a simplified version of the learning rule which uses both Hebbian and non-Hebbian plasticity rules to store memories, and evaluate its performance in comparison to classical Hopfield networks. Adding increasing biological realism to our model, we find that meta-learning of the plasticity recovers rules similar to the simplified model. We finally show how the feedforward, slot-based structure of our network allows it be naturally applied to more biologically-motivated memory tasks. Thus, the addition of our learning rules makes key-value memory compatible with applications of the classical Hopfield network, particularly as a biologically plausible mechanistic model of long-term memory in neuroscience.

3.2 Simplified learning mechanism

We consider a neuronal implementation of slot-based key-value memory, and endow it with a biologically plausible plasticity rule for memorizing inputs. We first describe a simplified learning rule for storing key-value pairs, which sets an upper bound on the network memory capacity and serves as a benchmark against existing memory networks. In later sections, we modify this rule to further increase biological realism.

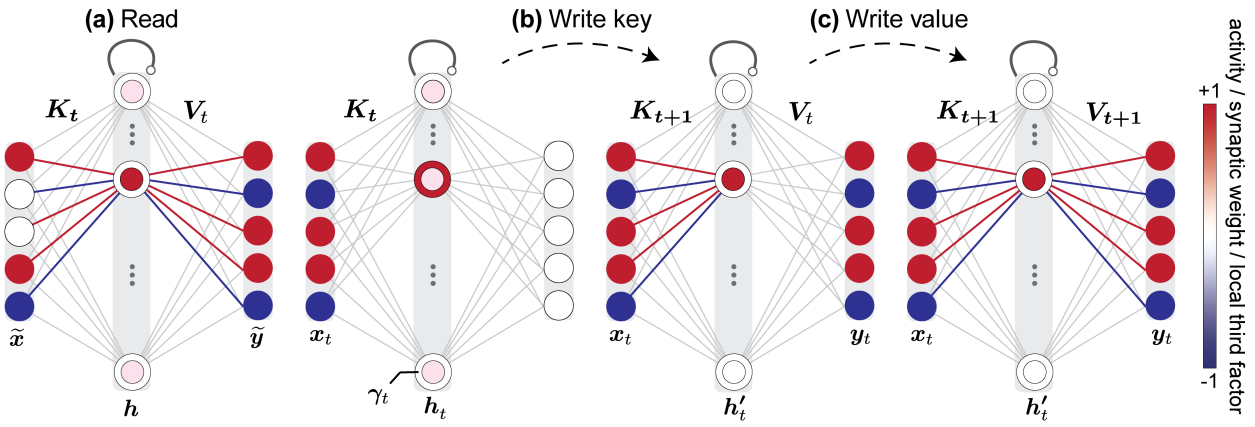


Figure 3.1: Network architecture and read/write mechanism. (a) Memory reading. The network is given a query $\widetilde{\boldsymbol{x}}$ (input layer activation with a corrupted version of stored key $\boldsymbol{x}$), selects the most similar stored key through approximately-one-hot hidden layer activity $\boldsymbol{h}$, and returns the corresponding value $\widetilde{\boldsymbol{y}} \approx \boldsymbol{y}$. (b) Writing keys. The input $\boldsymbol{x_t}$ is written into the i^{th} slot of the input-to-hidden weight matrix $\boldsymbol{K_t}$ by a "pre-only" plasticity rule, selecting the corresponding hidden neuron via local third factor $[\boldsymbol{\gamma_t}]_i = 1$. (c) Writing values. The same i^{th} hidden neuron is selected through an intermediate hidden unit activation $\boldsymbol{h'_t}$ and the target $\boldsymbol{y_t}$ is written to the hidden-to-output weight matrix $\boldsymbol{V_t}$ by a Hebbian update.

The network operates sequentially and consists of three fully-connected layers of neurons (Figure 3.1): a d-dimensional input with activity at time t given by the vector $\boldsymbol{x}_t$, an N-dimensional hidden layer $\boldsymbol{h}_t$, and an m-dimensional output layer $\boldsymbol{y}_t$. The i^{th} key memory slot corresponds to the synaptic weights from the input layer to a single hidden neuron (i^{th} row of the key matrix $\boldsymbol{K}_t$, storing key $\boldsymbol{x}_i$). The corresponding value is stored in the weights from that hidden neuron to the output (i^{th} column of the value matrix $\boldsymbol{V}_t$, storing value $\boldsymbol{y}_i$).

3.2.1 Reading

The network stores a set of key-value pairs $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, such that if a query $\widetilde{\boldsymbol{x}}$, e.g. a corrupted version of a stored key $\boldsymbol{x}$, is presented to the network, it returns the corresponding value $\boldsymbol{y}$. Given a query $\widetilde{\boldsymbol{x}}$ (Figure 3.1a), the hidden layer computes its similarity h_i to each stored key $\boldsymbol{x}_i$ (i^{th} row of $\boldsymbol{K}_t$) as a normalized dot product:

$$\boldsymbol{h} = \text{softmax}(\boldsymbol{K}_t \widetilde{\boldsymbol{x}}) \tag{3.4}$$

where the softmax function normalizes the hidden unit activations, and can be approximated biologically with inhibitory recurrent connections. The output layer then uses these similarity scores to compute the estimated value as the weighted sum of the stored values through a simple linear readout:

$$\widetilde{\boldsymbol{y}} = \boldsymbol{V}_t \boldsymbol{h} = \sum_{i=1}^{N} h_i \boldsymbol{y}_i \tag{3.5}$$

Assuming uncorrelated keys, the dot product of the query $\widetilde{\boldsymbol{x}}$ will be maximal with its corresponding stored key $\boldsymbol{x}$, and near-zero for all other stored keys, so $\boldsymbol{h}$ will be approximately one-hot. Thus, Eq. 3.5 reduces to $\widetilde{\boldsymbol{y}} \approx \boldsymbol{y}$ as desired. If target values are binary, $\boldsymbol{y}_t \in \{+1, -1\}^m$ (neurons are either active or silent), we use $\text{sign}(\widetilde{\boldsymbol{y}})$ when evaluating performance.

Mathematically, this architecture is equivalent to an MHN constrained to the heteroassociative setting with a fixed choice of activation functions [203], and recurrent dynamics updated for exactly one step [202]. Alternatively, it can be thought of as an differentiable version of an SDM [73] with a softmax rather than step function activation function in the hidden layer [60]. Unlike these networks, however, we introduce a novel biologically plausible writing mechanism to one-shot memorize key-value pairs by updating both the key and value matrices.

3.2.2 Writing keys

Key-value pairs are learned sequentially. Given an input $\boldsymbol{x}_t$ at time t, we write it into slot i of the key matrix using a non-Hebbian plasticity rule, where the presynaptic neuronal activity alone dictates the synaptic update, rather than pre- and postsynaptic co-activation as in a traditional Hebbian rule. A local third factor $[\boldsymbol{\gamma}_t]_i \in \{0, 1\}$ (Figure 3.1b, red circle) gates the plasticity of all input connections to hidden unit i, enabling selection of a single neuron for writing. Biologically, this may correspond to a dentritic spike [206, 207], which occur largely independently of the somatic

activity $\boldsymbol{h}_t$ performing feedforward computation[1]. This plasticity rule resembles behavioral time scale plasticity (BTSP) [209], recently discovered in hippocampus, a brain structure critical for formation of long-term memory.

In this simplified version we approximate local third factors as occurring in the least-recently-used neuron by cycling through the hidden units sequentially:

$$[\boldsymbol{\gamma}_t]_i = \begin{cases} 1 \text{ if } t = i \bmod N \\ 0 \text{ otherwise} \end{cases} \tag{3.6}$$

This can be biologically justified in several ways. Each neuron may have an internal timing mechanism that deploys a local third factor every N timesteps; the neurons may be wired such that a dendritic spike in neuron i primes neuron $i+1$ for a dendritic spike at the next time step; or the local third factors may be controlled by an external circuit that coordinates their firing. Alternatively, we consider a simpler mechanism, not requiring any coordination among the hidden layer neurons: each neuron independently has some probability p of generating a dendritic spike:

$$[\boldsymbol{\gamma}_t]_i \sim \text{Bernoulli}(p) \tag{3.7}$$

To gate whether a stimulus should be stored at all, we include a scalar global third factor $q_t \in \{0, 1\}$. Biologically, this may correspond to a neuromodulator such as acetylcholine [210] that affects the entire population of neurons, controlled by novelty, attention, or other global signals. Although it can also be computed by an external circuit, in our experiments this value is provided as part of the input. Thus, the learning rate of the synapse between input unit j and hidden unit i is the

[1]The same synaptic update could be accomplished without third factors, using a Hebbian rule, if the hidden layer were one-hot. However, since the hidden layer activity is determined by its weight matrix and the input, there is no simple neuronal mechanism to independently select a hidden neuron. More complicated versions may involve an external circuit which controls the activity of the hidden layer during writing, or strong feedforward weights which do not undergo plasticity [208].

product of the local and global third factors:

$$[\boldsymbol{\eta}_t^{\text{K}}]_{ij} = q_t[\boldsymbol{\gamma}_t]_i \tag{3.8}$$

Finally, to allow reuse of memory slots, we introduce a forgetting mechanism, corresponding to a rapid synaptic decay mediated by the local third factor. Whenever a synapse gets updated we have $[\boldsymbol{\eta}_t^{\text{K}}]_{ij} = 1$, so we can zero out its value by multiplying it by $1 - [\boldsymbol{\eta}_t^{\text{K}}]_{ij}$. If there is no update (either third factor is zero), the weight is not affected. The synaptic update is therefore:

$$\boldsymbol{K}_{t+1} = (1 - \boldsymbol{\eta}_t^{\text{K}}) \odot \boldsymbol{K}_t + \boldsymbol{\eta}_t^{\text{K}} \odot [\mathbf{1}\boldsymbol{x}_t^T] \tag{3.9}$$

where $\odot$ indicates the Hadamard (element-wise) product, and $\mathbf{1} \equiv (1, 1, .., 1)$.

3.2.3 Writing values

Having stored the key $\boldsymbol{x}_t$, the hidden layer activity at this intermediate stage is given by:

$$\boldsymbol{h}'_t = \text{softmax}(\boldsymbol{K}_{t+1}\boldsymbol{x}_t) \tag{3.10}$$

Note we are using the *updated* key matrix with $\boldsymbol{x}_t$ stored in the i^{th} slot. In the idealized case of random uncorrelated binary random memories $\boldsymbol{x}_t \in \{+1, -1\}^d$, the i^{th} entry of $\boldsymbol{K}_{t+1}\boldsymbol{x}_t$ will be equal to $\boldsymbol{x}_t \cdot \boldsymbol{x}_t = d$. All other entries will be near 0, since they are uncorrelated with $\boldsymbol{x}_t$. Thus, after normalization via the softmax, $\boldsymbol{h}'_t$ will be approximately one-hot, with $[\boldsymbol{h}'_t]_i \approx 1$.

To store the value, the output layer activity is clamped to the target $\boldsymbol{y}_t$ (Figure 3.1c). Biologically, this can be achieved either by strong one-to-one residual connections from the input to the output layer, or by having a common circuit which drives activity in both the input and output layers. Since the only hidden unit active is the one indexing the ith column of the value matrix, we can update the synapse between hidden unit i and output unit k by a Hebbian rule with learning rate $[\boldsymbol{\eta}_t^{\text{V}}]_{ki} = q_t[\boldsymbol{\gamma}_t]_i$

and rapid decay rate analogous to the key matrix:[2]

$$\boldsymbol{V}_{t+1} = (1 - \boldsymbol{\eta}_t^{\mathrm{v}}) \odot \boldsymbol{V}_t + \boldsymbol{\eta}_t^{\mathrm{v}} \odot \boldsymbol{y}_t (\boldsymbol{h}_t')^T \tag{3.11}$$

3.3 Results

3.3.1 Benchmark: autoassociative recall

As a simple evaluation of our algorithm's performance and comparison to other memory networks, we consider the classical autoassociative memory recall task where the key is equal to the value, ($d = m$, Figure 3.1). The network stores a set of T stimuli $\{\boldsymbol{x}_t\}$ and the query $\widetilde{\boldsymbol{x}}$ is a corrupted version of a stored key (60% of the entries are randomly set to zero). The network returns the corresponding uncorrupted version (section 3.5, Figure 3.6).

We compute the accuracy as a function of the number of stored stimuli (Figure 3.2a). By design, the plasticity rule with sequential local third factors performs identically to a simple non-biological key-value memory (TVT, [200], section 3.6). It has perfect accuracy for $T \leq N$ since every pattern is stored in a unique slot. For $T > N$, previously stored patterns are overwritten one-by-one and accuracy smoothly degrades. With random local third factors, accuracy degrades sooner due to random overwriting. Importantly, this holds for arbitrary network sizes, unlike the classical Hopfield network, which experiences a "blackout catastrophe" where all stored patterns are erased if the network size is large and the number of inputs exceeds its capacity [211, 212] (we do not see this here due to a relatively small network size).

Next, by measuring the number of patterns that the network can store before its recall accuracy drops below a threshold ($\theta = 0.98$), we can estimate this network's memory capacity C. This is an imperfect measure because some networks may have accuracy that stays above threshold longer but drops sharply after that. Nevertheless, this metric allows us to investigate empirically how the network's performance scales with its size (Figure 3.2b). The empirical scaling of the classical

[2] If local third factors are dendritic spikes in the *dendrites* of hidden layer neurons, it may not be realistic for the *axons* of the same neurons to be affected by these dendritic spikes. We consider this case for simplicity, as an upper-bound on performance. We will later lift this assumption without significantly hurting performance.

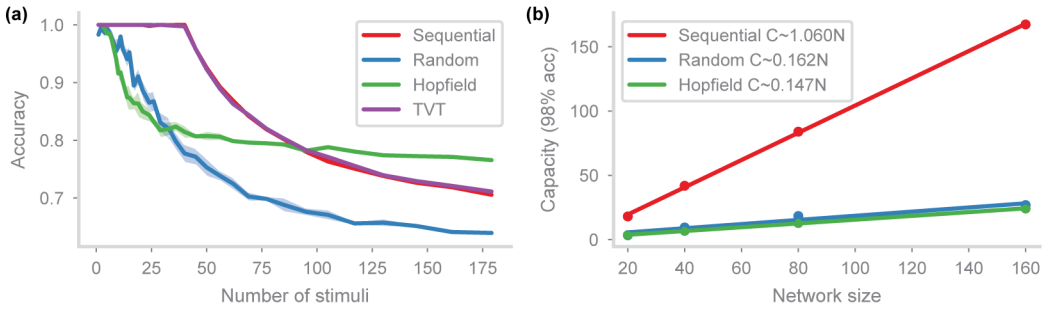


Figure 3.2: Our network ($d = N = 40$) with sequential and random ($p = 0.1$) local third factors, Hopfield network, and TVT [200] performance on the autoassociative memory task. (a) Accuracy as a function of stored stimuli. (b) Capacity (maximum number of stored stimuli at $\geq 98\%$ accuracy) as a function of network size.

Hopfield network is $C \sim 0.14N$, consistent with theoretical calculations [213], and the sequential algorithm's capacity scales approximately as $C \sim 1.0N$, as expected analytically. Importantly, the random algorithm's capacity also scales linearly, $C \sim 0.16N$, with a slope similar to the Hopfield network. Note, however, that with the same number of hidden neurons our network has twice as many connections as the Hopfield network. We also note that although the theoretical capacity of the Hopfield network can scale as $C \sim 2N$ [214, 215], to our knowledge there is not a learning algorithm that achieves this bound. Other work on autoassociative memory shows exponential scaling, but lacks a biologically plausible readout [216].

3.3.2 Meta-learning of plasticity rules

We now introduce parameters that can be meta-learned to optimize performance on a particular dataset without sacrificing biological plausibility. First, we enable varying the scale of the synaptic plasticity rates – each one is multiplied by a learnable parameter $\widetilde{\boldsymbol{\eta}}^{\text{K}}, \widetilde{\boldsymbol{\eta}}^{\text{V}}$:

$$[\boldsymbol{\eta}_t^{\text{K}}]_{ij} = q_t[\boldsymbol{\gamma}_t]_i[\widetilde{\boldsymbol{\eta}}^{\text{K}}]_{ij} \text{ and } [\boldsymbol{\eta}_t^{\text{V}}]_{ki} = q_t[\widetilde{\boldsymbol{\eta}}^{\text{V}}]_{ki} \tag{3.12}$$

Next, as a further improvement on biological plausibility, we remove the assumption that the hidden-to-output synapses (hidden layer axons) have access to the local third factors (hidden layer

dendritic spikes). Note that $\boldsymbol{\eta}_t^{\text{v}}$ above no longer contains a $\boldsymbol{\gamma}_t$ term. Instead, as a forgetting mechanism, the hidden-to-output synapses decay by a fixed factor $\widetilde{\boldsymbol{\lambda}}$ each time a stimulus is stored:

$$\boldsymbol{\lambda}_t = (1 - q_t) + q_t \widetilde{\boldsymbol{\lambda}} \tag{3.13}$$

where $\boldsymbol{\lambda}_t$ is the decay at time t, and $q_t \in \{0, 1\}$. Although in general $\widetilde{\boldsymbol{\eta}}^{\text{K}}, \widetilde{\boldsymbol{\eta}}^{\text{v}}$, and $\widetilde{\boldsymbol{\lambda}}$ can each be a matrix which sets a learning/decay rate for each synapse or neuron, we consider the simpler case where each is a scalar, shared across all synapses.

Most importantly, we parameterize the update rule itself. Each of the pre- and postsynaptic firing rates is linearly transformed before the synaptic weight is updated according to the prototypical "pre-times-post" rule. The jth input neuron's transformed firing rate is given by

$$[f^{\text{K}}(\boldsymbol{x})]_j = \widetilde{a}^{f^{\text{K}}} x_j + \widetilde{b}^{f^{\text{K}}} \tag{3.14}$$

and others $f^{\text{v}}, g^{\text{K}}, g^{\text{v}}$ are analogous. Although these functions can take arbitrary forms in general, this simple parameterization enables interpolating between traditional Hebbian, anti-Hebbian, and non-Hebbian (pre- or post-only) rules. The update rules can be summarized as follows:

$$\boldsymbol{K}_{t+1} = (1 - \boldsymbol{\eta}_t^{\text{K}}) \odot \boldsymbol{K}_t + \boldsymbol{\eta}_t^{\text{K}} \odot [g^{\text{K}}(\boldsymbol{h}_t) f^{\text{K}}(\boldsymbol{x}_t)^T] \tag{3.15}$$

$$\boldsymbol{V}_{t+1} = \boldsymbol{\lambda}_t \odot \boldsymbol{V}_t + q_t \widetilde{\boldsymbol{\eta}}^{\text{v}} \odot [g^{\text{v}}(\boldsymbol{y}_t) f^{\text{v}}(\boldsymbol{h}'_t)^T] \tag{3.16}$$

We begin by training this network on the benchmark task from Section 3.3.1, optimizing these parameters using stochastic gradient descent (Adam, [217]). Figure 3.3a shows the performance of the meta-learned algorithms in comparison to the corresponding simplified versions (section 3.2) for either sequential and random local third factors. Importantly, removing the unrealistic local-third-factor-mediated rapid synaptic decay in the hidden-to-output synapses and replacing it with a passive decay does not significantly impact performance, as long as the decay rate is appropriately set. Indeed, this even slightly improves performance for longer sequences.

In the case of a random local third factor, we also compare values of p (not trained). Empirically, we find that $p \approx 4/N$ produces desirable performance: it ensures that the probability of no local third factor occurring (and therefore no storage) is small ($< 2\%$) while minimizing overwriting. Computing the capacity (Figure 3.3b), we see that there is an advantage for the probability to scale with the network size ($p = 4/N$) rather being a fixed value ($p = 0.1$).

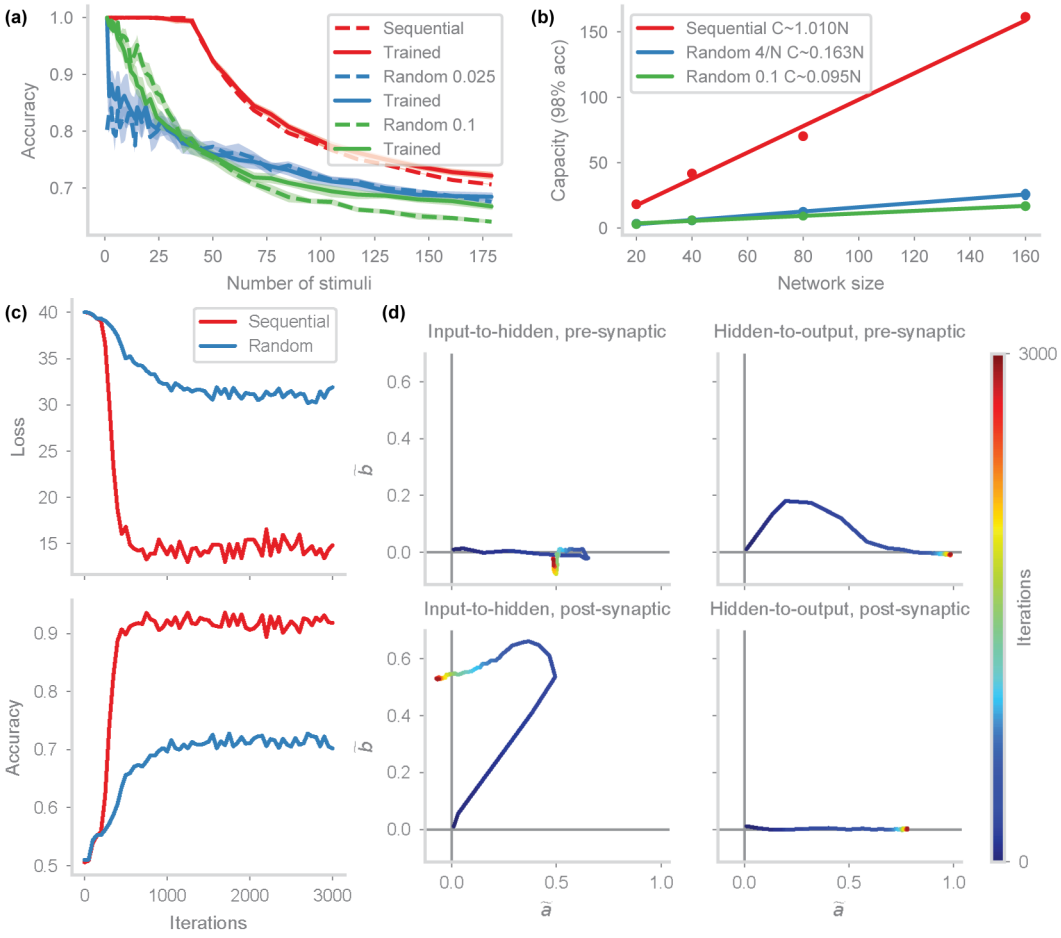


Figure 3.3: (a) Performance comparison of simple (dashed curves) and meta-learned (solid) network ($d = N = 40$, p varied, shown in legend.) (b) Capacity of trained network with sequential or random local third factor. In the random case, p is either fixed or scales with N. (c) Training the sequential and random ($p = 0.1$) network from (a,b). Although accuracy is not 1.0 during training, the network successfully generalizes to unseen sequence lengths. (d) Learning trajectory of the plasticity parameters in the sequential network converges to qualitatively similar solutions as the original simplified network (see also Figure 3.9). Note that the input-to-hidden plasticity is independent of the post-synaptic firing rate (bottom left plot, $\widetilde{a}^{g^{\kappa}} \approx 0$)

We next investigate the training trajectories. Figure 3.3c shows the loss and accuracy curves

over the course of training for networks with sequential and random local third factors. Training data consists of sequence lengths between $T = N/2$ and $T = 2N$ (above capacity for both versions of the network), so accuracy does not reach 1.0. Nevertheless, the network successfully generalizes to lengths outside of this range, indicating a robust mechanism for memory storage.

Most importantly, we examine the plasticity rule discovered by optimization. Figure 3.3d shows the slope $\widetilde{a}$ and offset $\widetilde{b}$ parameters for the firing rate transfer functions $g^{\text{K}}, f^{\text{K}}, g^{\text{V}}, f^{\text{V}}$ in the sequential algorithm over the course of meta-learning. The plasticity rule for the input-to-hidden connections (Figure 3.3d, left) becomes pre-dependent ($\widetilde{a}^{f^{\text{K}}} \approx 0.5$, $\widetilde{b}^{f^{\text{K}}} \approx 0$) but not post-dependent ($\widetilde{a}^{g^{\text{K}}} \approx 0$, $\widetilde{b}^{g^{\text{K}}} \approx 0.5$), analogous to the idealized plasticity rule described in section 3.2. Similarly, the plasticity rule for the hidden-to-output connections becomes Hebbian (both pre-dependent and post-dependent, $\widetilde{b}^{g^{\text{V}}} \approx \widetilde{b}^{f^{\text{V}}} \approx 0$, but $\widetilde{a}^{g^{\text{V}}} \approx \widetilde{a}^{f^{\text{V}}} \approx 1$). The random version shows qualitatively similar trained behavior (Figure 3.9).

Since the performance and parameterization of the meta-learned algorithm is almost identical to the simple case when using sequential local third factors, we only consider the random version for meta-learning in subsequent sections.

3.3.3 Continual, flashbulb, and correlated memory tasks

We next test our plasticity rules on more ecologically relevant memory tasks. These tasks reflect the complexity of biological stimuli and the functionality needed for versatile memory storage.

In realistic scenarios, memories are stored and recalled in a continual manner – the subject sees an ongoing stream of inputs and is required to recall a stimulus that was shown some time ago (section 3.5) – rather than being presented a full dataset to memorize before testing, as in the benchmark autoassociative task. Figure 3.7a shows such a sample dataset with a delay interval of 2. Our learning algorithm is naturally suited for this task due to its decay mechanisms – recent stimuli are stored, but older ones are forgotten. Its accuracy decreases in steps of width N due to the sequential nature of the local third factor (section 3.5). Performance with random third factors decays smoothly since writing is stochastic, but both networks show better overall performance than

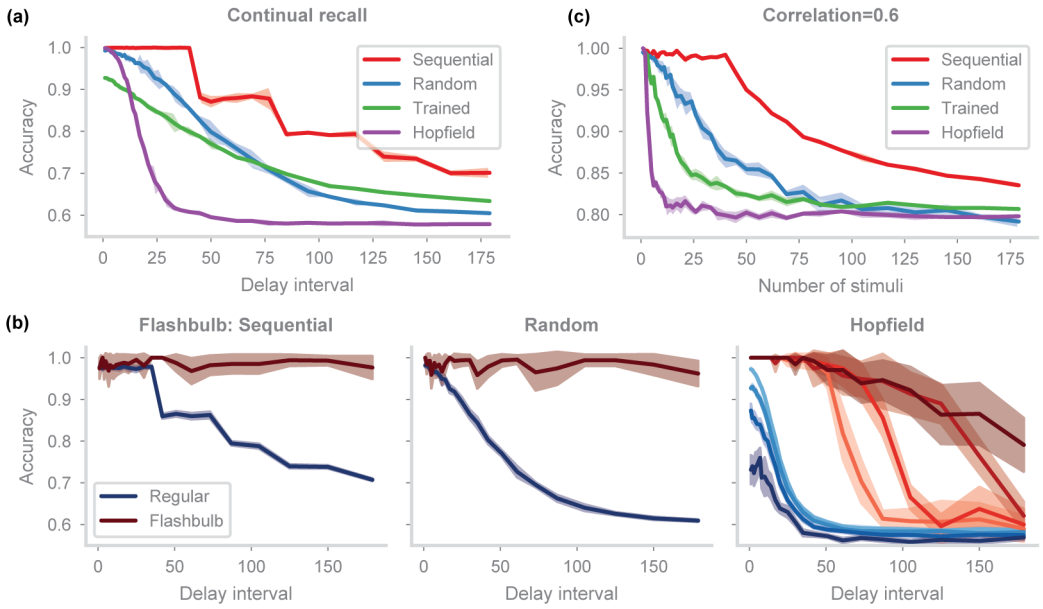


Figure 3.4: (a) Performance on continual recall task, accuracy measured as a function of the number of timesteps between the storage and recall of a memory. "Trained" corresponds to the meta-learned algorithm with random local third factors. (b) Same as (a), but including five "flasbulb" memories. Performance of simplified algorithm with meta-plasticity, using sequential (left) or random (middle) local third factors. For Hopfield network (right), increasing darkness of each line corresponds to higher write strength for flashbulb memories, with values 10, 50, 10^3, and 10^6. (c) Same as Figure 3.2a, but with memories having correlation of 0.6.

the Hopfield network (Figure 3.4a).[3]

Next, we consider "flashbulb" memories, a phenomenon where highly salient experiences are remembered with extreme efficacy, often for life [219]. Functionally, these memories may be used to avoid adverse experiences or seek out rewarding ones. We modify the continual recall task such that a small number of stimuli are deemed salient and accompanied by a stronger global third factor ($q_t = 10$) akin to a boosted neuromodulatory influence on learning (Figure 3.7b). We add a simple meta-plasticity mechanism to our plasticity rule, a stability parameter $[\boldsymbol{S}_t]_{ij}$ for each synapse, initially set to 0. If the learning rate for that synapse crosses a threshold $[\boldsymbol{\eta}_t]_{ij} > 1$, then $[\boldsymbol{S}_{t+k}]_{ij} = 1$ for all $k > 0$ and suppresses subsequent plasticity events. Thus, the learning rate for the key matrix

[3]For fair comparison, we introduce an empirically chosen synaptic decay parameter $\lambda = 0.95$ to the Hopfield network. We also considered a version of the Hopfield network with bounded weights [218], designed for continual learning, but found that synaptic weight decay has better overall performance. Furthermore, we include a global third factor which controls overall plasticity identically to our model.

is as follows (learning rate $\boldsymbol{\eta}_t^{\text{v}}$ is analogous):

$$[\boldsymbol{\eta}_t^{\text{k}}]_{ij} = (1 - [\boldsymbol{S}_t^{\text{k}}]_{ij}) q_t [\boldsymbol{\gamma}_t]_i \tag{3.17}$$

With this meta-plasticity, the network retains flashbulb memories with minimal effect on its recall performance on regular memory items (Figure 3.4b, left, middle). To perform this task in the Hopfield network, introducing synaptic stability is significantly detrimental to performance, so we simply store flashbulb memories as large-magnitude updates to the weight matrix. As a result, it exhibits a tradeoff between storage of regular items and efficacy of flashbulb memories (Figure 3.4b, right). Thus, an important benefit of the slot-based storage scheme is that flexible treatment of individual memories can be naturally implemented.

Finally, real-world stimuli are often spatially and temporally correlated – for instance, two adjacent video frames are almost identical. Storing such patterns in a Hopfield network causes interference between attractors in the energy function, decreasing its capacity. In contrast, by storing stimuli in distinct slots of the key and value matrices, key-value memory can more easily differentiate between correlated but distinct memories. To verify this, we use a correlated dataset by starting with a template vector and generating stimuli by randomly flipping some fraction of its entries (Figure 3.7c). The performance of the plasticity rule is similar to that for uncorrelated data (Figure 3.2a), with minor degradation due to spurious recall of similar stored memories (Figure 3.4c). Figure 3.10 shows similar results for varying correlation strengths.

3.3.4 Heteroassociative and sequence memory

The network and learning mechanism is agnostic to the relationship between the input and target patterns, and so naturally generalizes to heteroassociative memory. To draw comparisons with Hopfield-type networks, we consider the Bidirectional Associative Memory (BAM) [220], a generalization of the Hopfield network designed for heteroassociative recall. We evaluate the networks on a modified version of the recall task from Section 3.3.1 where values are distinct from

keys, $\boldsymbol{y}_t \in \{+1, -1\}^m$ for $d \neq m$ (Figure 3.5a, top; Figure 3.8). Compared to the autoassociative task (Figure 3.2), the Hopfield-type network's performance on the heteroassociative task deteriorates (Figure 3.5, bottom). On the other hand, the performance of our network remains unaffected in the heteroassociative task, as its factorized key-value structure allows more flexibility in the types of memory associations learned.

A more biologically relevant version of heteroassociative memory is sequence learning. Experimental and theoretical evidence suggests that hippocampal memory can serve as a substrate for planning and decision making through the replay of sequential experiences [221, 222, 223]. As a simple probe for a similar functionality, we use a sequence recall task where the network is presented with a sequence of patterns to memorize, using the value at time t as the key at time $t + 1$. Afterwards, it is prompted with a random pattern from the sequence and tasked with recalling the rest. By adding a recurrent loop, using the output $\widetilde{\boldsymbol{y}}_t$ at time t as the input $\boldsymbol{x}_{t+1}$ at the next timestep (Figure 3.5b, top), our network with sequential local third factors can perform sequence learning without error until its capacity limit (Figure 3.5b, bottom). BAM does not perform as well, likely due to propagating errors from incorrect recall of earlier patterns in the sequence.

Finally, we consider a more complex task that requires our memory module to be integrated within a larger system, demonstrating the modular nature of our memory network. In the "copy-paste" task [126], the system must store a variable-length sequence of patterns $((\boldsymbol{s_1}, \boldsymbol{s_2}, \ldots, \boldsymbol{s_{T+1}}))$ and output this sequence when the end-of-sequence marker is seen (Figure 3.8). We train an external network as a "controller" to generate $\boldsymbol{x}_t$, $\boldsymbol{y}_t$, and $\boldsymbol{q}_t$ (Figure 3.5c, top) (for BAM, $\boldsymbol{q}_t$ scales the magnitude of the Hebbian update). With this controller, our network with sequential local third factors successfully learns the task and generalizes outside the sequence lengths seen (Figure 3.5c, bottom). The random and trained networks do not perform as well, likely due to lower memory capacity. BAM successfully learns the task, but is not able to generalize outside the sequence lengths seen in training.

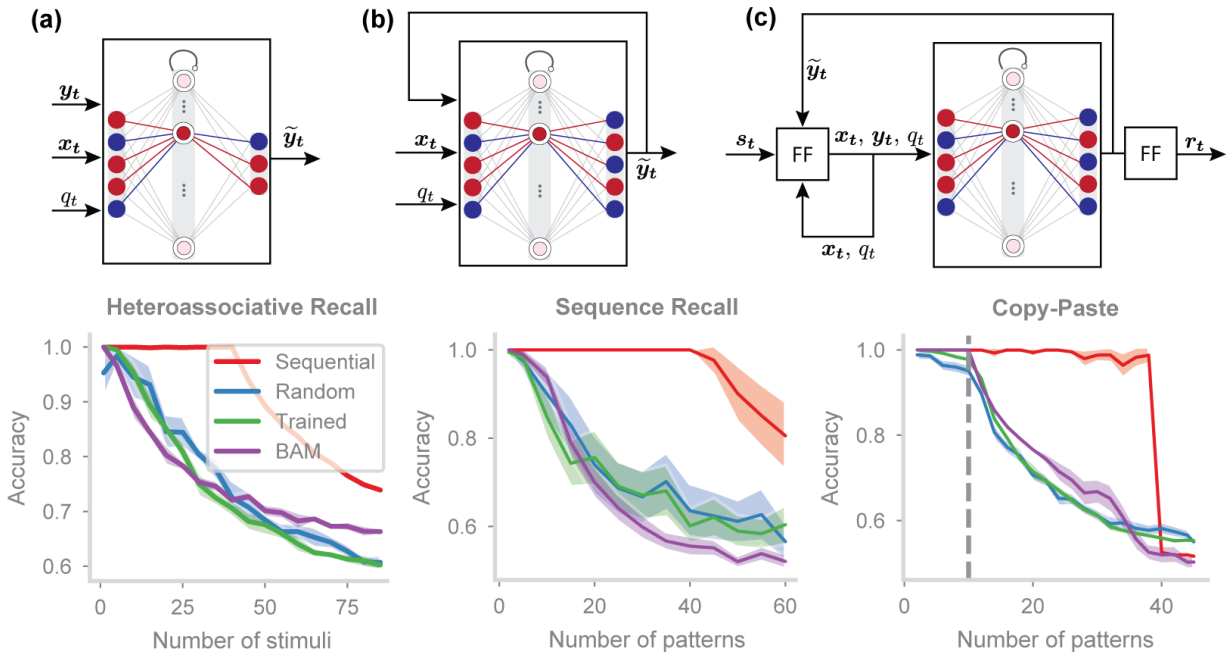


Figure 3.5: (a) Architecture ($d = N = 40, m = 20$) and performance on heteroassociative version of the benchmark task (Figure 3.2). (b) Recurrent architecture modification for sequence recall ($d = N = 40, m = 40$), $\boldsymbol{x}_{t+1} = \widetilde{\boldsymbol{y}}_t$. Accuracy is plotted as a function of the length of the entire sequence presented. (c) Network is embedded in a larger system with a feedforward network to perform the copy-paste task ($d = N = 40, m = 40$). Dashed line is the maximum number of patterns shown during training.

3.4 Discussion

We proposed models of biological memory by taking inspiration from key-value memory networks used in machine learning. These biologically plausible models use a combination of Hebbian and non-Hebbian three-factor plasticity rules to approximate key-value memory networks. Due to the flexibility of their structure, they can naturally be adapted for biologically relevant tasks. Our results suggest an alternative framework of biological long-term memory that focuses on feedforward computation, as opposed to recurrent attractor dynamics. Importantly, both our hand-designed and meta-learned results propose a role for recently discovered non-classical plasticity rules [209] that are advantageous for this type of feedforward computation. Furthermore, we propose an architecture where individual memories are stored more independently of each other than in Hopfield networks. Such a factorized design creates opportunities for more versatile use and control

of memory.

Several questions remain. First, although long-term memory describes many categories of memory supported by various brain regions, we have not disambiguated these differences and their implications on testing and interpreting our model. To validate our network as not merely a plausible model but as a true model of the brain, it is critical to make direct comparisons between our algorithm and experimental findings in memory-related behavior and neural activity. Furthermore, our aim is not to be competitive with state-of-the-art memory networks but rather to provide a biologically realistic learning algorithm for MHNs that is on par with classical Hopfield networks. To this end, we focus on artificial stimuli with simple statistical structures; it remains unclear how our models will perform with more complex and naturalistic data, or how they compare to their non-biological counterparts.

Taken together, our results take a neuroscience-minded approach to connect two lines of work in memory networks that have been largely disparate.

3.5 Task details

3.5.1 Benchmark: autoassociative recall

For the autoassociative memory benchmark task, we generate T memories, each of which is a uniformly randomly generated d-dimensional vector $\boldsymbol{x}_t \in \{+1, -1\}^d$, for $t = 1, \ldots, T$. During the storage phase, the key and value matrices are initialized to zero and each pattern is shown to the network sequentially with the network's global third factor $q_t = 1$ active for all patterns. For storage, both the network's input and output layers are clamped to the input value $\boldsymbol{x}_t$. Next, during the test phase, the network is shown queries $\widetilde{\boldsymbol{x}}_t$, corresponding to the previously shown stimuli with 60% of the entries in each vector randomly set to zero (Figure 3.6). Queries are shown in the same order as the stimuli and the global third factor $q_t = 0$ to ensure no plasticity occurs. Only the input layer is clamped and the result is read out from the output layer $\widetilde{\boldsymbol{y}}_t$. Accuracy is computed as the

total fraction of correctly recalled entries, calculated for varying values of T:

$$\text{accuracy} = \frac{1}{Td} \sum_{t=1}^{T} \sum_{i=1}^{d} \mathbb{I}\left\{[\boldsymbol{x}_t]_i = \text{sign}([\widetilde{\boldsymbol{y}}_t]_i)\right\} \tag{3.18}$$

Note that for the classical Hopfield network, the input and readout neurons are the same, so by presenting a query $\widetilde{\boldsymbol{x}}_t$, a fraction of the output bits in $\widetilde{\boldsymbol{y}}_t$ are *a priori* set to the correct values from $\boldsymbol{x}_t$. This raises the chance level of the classical Hopfield network compared to the other networks we consider in Figure 3.2a.

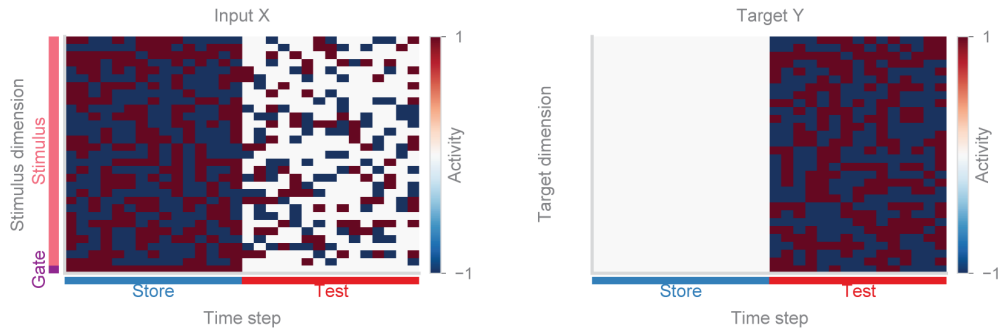


Figure 3.6: Autoassociative recall benchmark task. $d = 30, T = 15, 60\%$ occluded during test.

3.5.2 Beyond simple recall

To test the network in a continual setting, rather than datasets of fixed length T, we use arbitrarily long datasets where the network is asked to recall a stimulus that was presented R timesteps ago. To generate the dataset, at each timestep with probability $p_{\text{gen}} = 0.5$ the input $\boldsymbol{x}_t$ is a randomly generated binary vector (as in the benchmark dataset). With probability $1 - p_{\text{gen}} = 0.5$, the input is a query (as in the benchmark dataset) $\widetilde{\boldsymbol{x}}_t$, corresponding to the input shown R timesteps ago[4], $\boldsymbol{x}_{t-R}$. For the generated stimuli which are subsequently queried, the modulator $q_t = 1$ during their initial presentation. Otherwise, $q_t = 0$. To ensure that the network is operating in steady state and therefore in the continual learning regime, we use a long trial duration $T = \max(1000, 20R)$. Figure 3.7a shows 30 timesteps of such a trial with $R = 2$. Note that in a single trial, the delay

[4]We furthermore ensure that a query is not presented twice, so if the input R timesteps ago was a query, a stimulus vector is generated.

interval between the stored stimulus and the query is always a fixed value R. However, we test the network on multiple trials, each with a different value of R.

Performance of the network with sequential local third factors decreases in steps of width N because it selects the next slot at each timestep regardless of whether the current one was written to. Since a global third factor does not occur at every timestep, some slots left untouched when the local third factor selects them for a second time, thus preserving their contents with a probability which depends on the frequency of queries in the stream. This probability is the same for all delays of length $N+1$ to $2N$, slightly lower for all delays of length $2N+1$ to $3N$ (i.e. the slot doesn't get written when the local factor selects it the second *and* the third time), and so forth, resulting in a stepwise curve.

The "flashbulb" memory task is similar to the continual task, however for every trial, 5 memories are selected as flashbulb memories. These are generated as the others, but are accompanied by a very strong modulatory input $q_t = 10$ rather than the normal $q_t = 1$. Figure 3.7b shows a portion of the continual stream, including two of the flashbulb memories.

To test the network performance on datasets with correlated stimuli, we generate T binary random vectors and evaluate the network as in the benchmark task (Figure 3.6). The first "template" vector is generated randomly $\boldsymbol{x}_1 \in \{+1, -1\}^d$ as before. All subsequent stimuli are generated by randomly flipping a fraction $(1-\rho)$ of the entries in the template vector, resulting in correlated stimuli with $\mathrm{corr}(\boldsymbol{x}_t, \boldsymbol{x}_{t'}) = \rho$ (Figure 3.7c). Figure 3.10 shows the network performance for additional values of ρ.

3.5.3 Beyond autoassociative memory

The heteroassociative recall task (Figure 3.7a) is identical to the autoassociative memory benchmark task (Figure 3.6) except we have the additional generation of m-dimensional vectors $\boldsymbol{y}_t \in \{+1, -1\}^d$, for $t = 1, \ldots, T$. For our task, $m = \frac{d}{2}$. Thus, although during the storage phase the network's input is clamped to some input value $\boldsymbol{x}_t$ as in the autoassociative benchmark, the output layer is clamped to the output value $\boldsymbol{y}_t$.

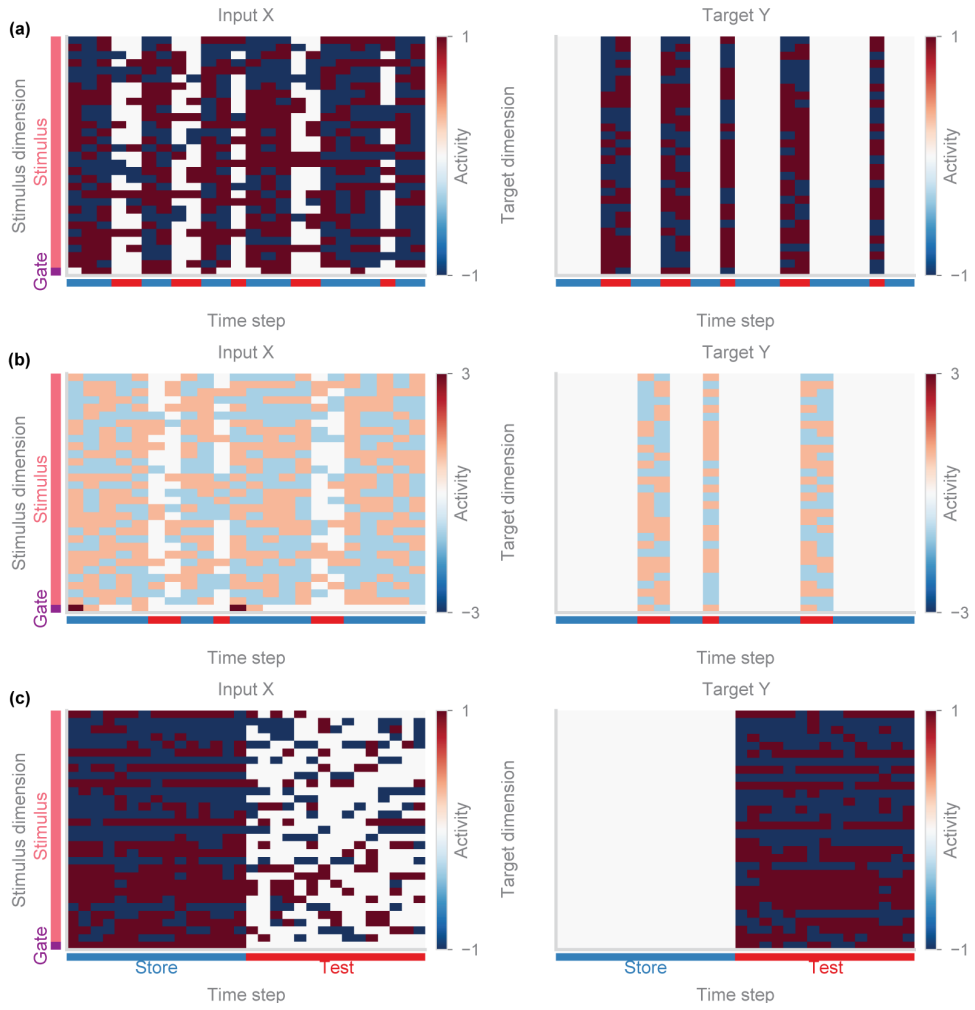


Figure 3.7: (a) Thirty timesteps of a continual autoassociative recall task with $R = 2$. (b) Thirty timesteps of a flashbulb task, showing two of the flashbulb memories. Note colorbar range. For visualization, modulation strength $q_t = 3$ during flashbulb memories. (c) Autoassociative recall task for correlated memories with $\rho = 0.6$.

In the sequence recall task (Figure 3.7b), similar to the benchmark task, we randomly generate T memories, each of d-dimensions. This forms a T-length sequence. During the testing phase, a prompt pattern $\boldsymbol{x}_t$ from the middle of this sequence is shown. The goal of the task is to then return the rest of the patterns in this sequence in order: $(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+2}, \ldots, \boldsymbol{x}_T)$. We run our network recurrently so that, at time t, we clamp the network input to $\text{sign}(\widetilde{\boldsymbol{y}}_{t-1})$ and the network output to $\boldsymbol{x}_t$.

In the copy-paste task (Figure 3.7c) we begin by randomly generating a T-length sequence as in the sequence recall task. Each pattern $\boldsymbol{s_t}$ is of dimension $D = 25$ (we use a different variable name since the stored keys $\boldsymbol{x_t}$ will be different than the elements of the sequence). We add an additional

dimension to each pattern and an additional pattern to the sequence, such that the sequence is $(D+1) \times (T+1)$. The additional dimension is used to denote the end-of-sequence (EOS) marker and is set to -1 when it is not in use. The EOS marker is shown at the end of the sequence, at time $T+1$. Thus, the vector shown to the network at time $T+1$ is $\boldsymbol{s}_{T+1} = [-1 \quad -1 \ldots \quad +1]$. After seeing the EOS marker, the goal of the task is to repeat the entire sequence $(\boldsymbol{s_1}, \boldsymbol{s_2}, \ldots, \boldsymbol{s_{T+1}})$. During training and evaluation, T is randomly drawn from 1 to 10 in the task.

We use three feedforward controller networks coupled with our memory network. At time t, each controller network receives a $(D+2d+1)$-dimensional input $\boldsymbol{v_t}$: a concatenation of $\boldsymbol{s_t}, \boldsymbol{x_{t-1}}, \widetilde{\boldsymbol{y}}_{\boldsymbol{t-1}}$ and q_{t-1}. The outputs for the three networks are the d-dimensional key $\boldsymbol{x_t}$ ($d = 40$), d-dimensional value $\boldsymbol{y_t}$, and scalar global third factor q_t for the memory module as follows. Then,

$$\boldsymbol{x'_t} = \tanh(\boldsymbol{R_x v_t} + \boldsymbol{b_x})$$

$$\boldsymbol{y'_t} = \tanh(\boldsymbol{R_y v_t} + \boldsymbol{b_y})$$

$$q_t = \sigma(\boldsymbol{R_q v_t} + \boldsymbol{b_q})$$

where $\sigma(\cdot)$ is the logistic function, and $\boldsymbol{R_x}$,$\boldsymbol{R_y}$,$\boldsymbol{R_q}$,$\boldsymbol{b_x}$,$\boldsymbol{b_y}$,$\boldsymbol{b_q}$ are learned matrices. These outputs are normalized to have L2-norm $\sqrt{d}$ to match the norm of the inputs presented in the autoassociative memory task:

$$\boldsymbol{x_t} = \sqrt{d}\frac{\boldsymbol{x'_t}}{||\boldsymbol{x'_t}||} \tag{3.19}$$

$$\boldsymbol{y_t} = \sqrt{d}\frac{\boldsymbol{y'_t}}{||\boldsymbol{y'_t}||} \tag{3.20}$$

The controller outputs $\boldsymbol{x_t}, \boldsymbol{y_t}, q_t$ are presented to the memory network, which is updated according to our proposed plasticity rules. With the updated key and value matrices, we retrieve the output of the memory module $\widetilde{\boldsymbol{y}}_{\boldsymbol{t}}$, using $\boldsymbol{x_t}$ as the query. Finally, $\widetilde{\boldsymbol{y}}_{\boldsymbol{t}}$ is fed into a one-layer network to

transform the output from d dimensions to a D-dimensional output $\boldsymbol{r_t}$,

$$\boldsymbol{r_t} = \tanh(\boldsymbol{R_o}\widetilde{\boldsymbol{y}}_{\boldsymbol{t}} + \boldsymbol{b_o}) \tag{3.21}$$

where $\boldsymbol{R_o}$ is learned. The values $\boldsymbol{x_t}, \widetilde{\boldsymbol{y}}_{\boldsymbol{t}}, q_t$ are then fed back into the controller as the input for the next time step, along with $\boldsymbol{s_{t+1}}$. The initial inputs $\boldsymbol{x_0}$, $\widetilde{\boldsymbol{y}}_{\boldsymbol{0}}$ and q_0 corresponding to $\boldsymbol{s_1}$ are also learned.

When the network is prompted with the EOS marker, the output $(\boldsymbol{r_{T+2}}, \ldots, \boldsymbol{r_{2T+2}})$ should be equal to the original sequence $(\boldsymbol{s_1}, \ldots, \boldsymbol{s_{T+1}})$. We train the network end-to-end with backpropagation through time to minimize mean squared error loss.

For our simulations with BAM, we follow the same controller set-up as above. As is the case for the previous heteroassociative tasks, we use the typical BAM update and update the weight matrix by $\eta \boldsymbol{x_t} \boldsymbol{y_t}^\intercal$ with learning rate η. The learning rate is modulated by the global third factor, as is the case for our models. However, for training purposes, we found it helpful to scale the learning rate such that $\eta = \frac{1}{40} q_t$.

3.6 TVT key-value memory mechanism

TVT is an algorithm that enhances the learning of memory-based agents by combining attentional memory access with reinforcement learning [200]. Here, we used the key-value memory mechanism used by the model where inputs were written to memory and attentional memory access was used to read stored inputs from memory. Unlike in the original work, there is no LSTM controller or reinforcement learning component. We simply use the read and write functions to a memory matrix as in the original paper, but do not use the TVT algorithm itself or any of the additional architecture used in the original authors' work.

First, a memory matrix is initialized whose rows will each store one stimulus along with its read strength. There is a reader network and a writer network for the read and write operations respectively. A call to write stores a stimulus. A call to read returns the H most similar memories,

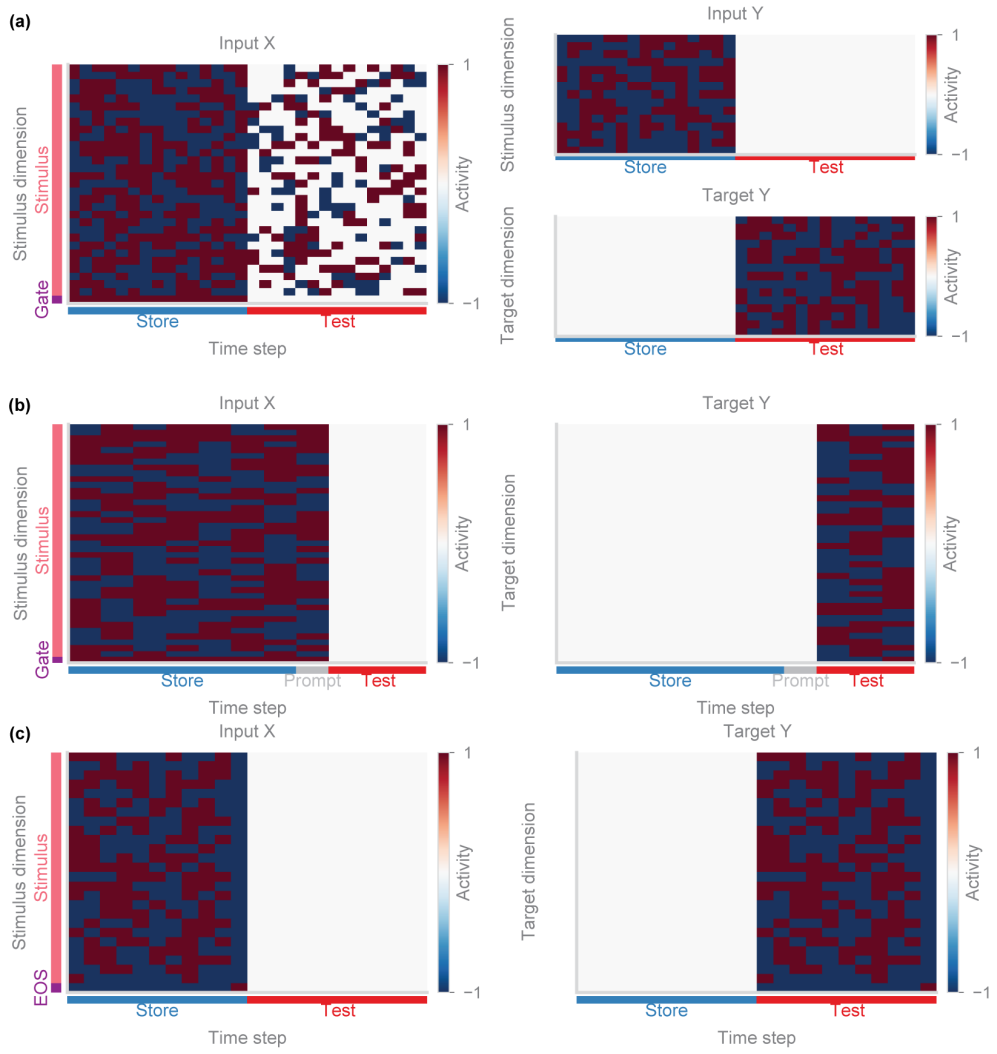


Figure 3.8: (a) Heteroassociative recall task. $d = 30, m = 15, T = 15, 60\%$ occluded during test. (b) Sequence recall task with $d = 40$ and $T = 7$. The prompt is the 4th pattern of the sequence. (c) Copy-paste task with $d = 8, T = 10$.

where H is the number of read heads, or locations that can be read from simultaneously.

For the recall task, the writer receives an index indicating which row in memory should be written to. During a write to memory, the specified row of the memory matrix is cleared and the input vector is written to this cleared slot. During the storage phase, input vectors are stored sequentially such that each incoming input vector is written to the next unfilled row of the memory matrix. If the memory matrix is full, a filled row beginning with the first row will be cleared and an incoming input will be stored in it.

During the retrieval phase of the recall task, the reader uses attentional memory access to retrieve a weighted version the H most similar (smallest in cosine distance) memories from the memory, using H read heads. First, it is given $M \times H$ tensor of inputs where M is the length of each input vector for each of the H read heads. Next, the read keys and the weights used for each key are computed by passing the input through a linear layer that produces an $(M + 1) \times H$ output. The softplus function is then applied to the keys and read strengths output by this linear layer. The resulting $(M + 1) \times H$ tensor is separated into a $M \times H$ tensor of read keys and a $H \times 1$ tensor of read strengths for each read head.

The read keys and the values in the memory matrix are then normalized and multiplied together. This yields a tensor of cosine distances between each read key and each item in memory.

This is multiplied by the $H \times 1$ tensor of read strengths, yielding a $H \times R$ tensor of weighted distances, where R is the number of rows in the memory matrix. These weighted distances are then passed through a softmax function. Each row then has one element (corresponding to one row in the memory) that is maximally activated. This tensor is then multiplied by the memory matrix, yielding a tensor of memory reads that is a weighted sum of the rows most similar to the weighted read keys.

The linear layer used to generate the keys and read strengths is learned using SGD. A model trained on 20000 steps was used, and with a 40-row memory matrix (to be compared with a size 40 hidden layer of our network).

In the simplified model, unlike in the original paper, only a single read head was used in order to make comparisons with our network. The TVT's key-value memory mechanism works very similarly to the our network with sequential local third factors. The sequential network, however, adds in the biological feature of plasticity rules to store memories.

3.7 Supplementary results

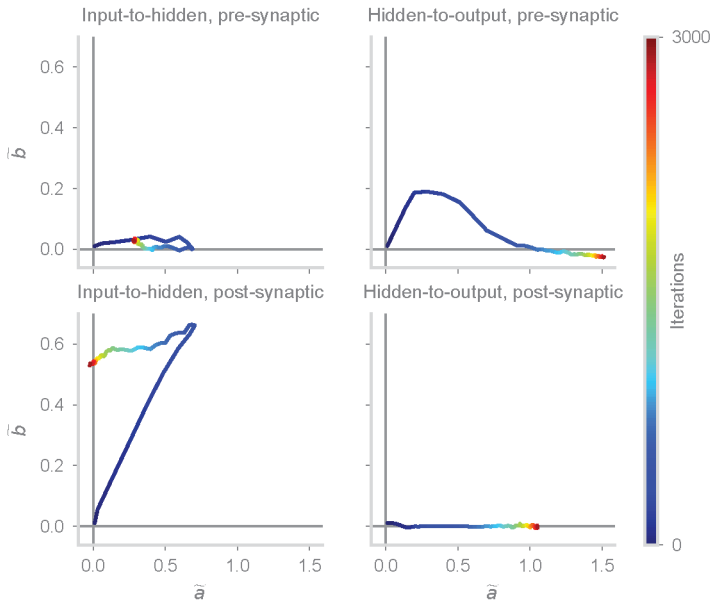


Figure 3.9: Same as Figure 3.3d, but for a network with random local third factors.

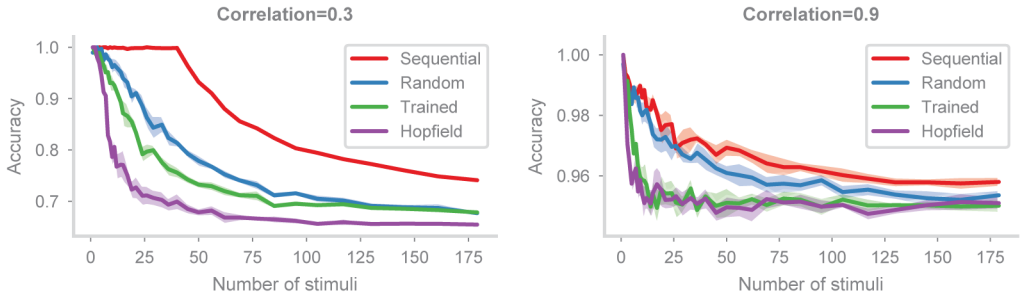


Figure 3.10: Same as Figure 3.4c, with different correlation strengths.

Chapter 4: Memorization and consolidation in associative memory networks

The work presented in this chapter was done in collaboration with Kimberly Stachenfeld, Dmitry Krotov, and LF Abbott. A subset of these results is published in [114]. We are grateful to John Cunningham, Ashok Litwin-Kumar, Stefano Fusi, and James Fitzgerald for helpful discussions and suggestions. Thanks to Jan Funke for suggesting importance sampling as a potential approach.

Humans, animals, and machines can store and retrieve long-term memories of individual items, while at the same time consolidating and learning general representations of categories, discarding the individual examples from which the representations were constructed. Classical neural networks, however, model only one of these two regimes. In this work, we propose a biologically motivated model that can not only consolidate representations of common items but also memorize exceptional ones. Critically, we consider the unsupervised learning regime where exceptional items are not labeled as such a priori, so the signal to either memorize or consolidate items must be extracted from the network itself. We propose a number of metrics for this signal and compare them for two different algorithms inspired by traditional imbalanced data learning approaches – loss reweighting and importance sampling. Overall, our model serves not only as a framework for concurrent memorization and consolidation processes in biological systems, but also as a simple illustration of related phenomena in large-scale machine learning models, as well as a potential method for debiasing artificial intelligence algorithms.

4.1 Introduction

In human and animal experience, long-term memories of individual items can coexist with generalized representations of categories. Individual items are, by definition, stored verbatim

(*memorization*), such that the specific details of the item can be recovered when needed. In other cases, it can be advantageous to blend multiple examples from a category to create a flexible representation, with the specifics of the individual examples being forgotten after *consolidation*; conversely, seeing many related examples can induce the creation of a generalized category. These processes are two essential but often opposing aspects of flexibly representing knowledge about the environment [224, 225].

There are a number of examples from the neuroscience literature demonstrating a capacity for both memorization and consolidation in humans and animals. For instance, human subjects were trained to extrapolate new partially-occluded trajectories of moving dots based on previously viewed example trajectories [226]. If trained on a few examples, subjects tend to memorize simple mappings from stimuli to trajectory endpoints, whereas a large diverse training set results in learning more generalizable predictive models. Other human studies investigate learning categories defined by a general rule plus exceptions (e.g., if the data point has feature X, it is category A, unless it also has feature Y), noting that such categories are readily learned, and that exceptions have neural representations different from the rule-following items [227, 228]. In a different domain, mice were subjected to a social recognition paradigm, given the option to explore a container with either a novel conspecific or a familiar littermate. Neural representations recorded from the CA2 region of the hippocampus favor either a high-dimensional geometry when interacting with familiar mice (optimal for memorization), and a low-dimensional geometry for novel mice (optimal for generalization) [229].

Classical models typically operate in only one of these modes. For example, Hopfield networks [230] can memorize and retrieve a predetermined set of datapoints, whereas deep networks [231] learn a compact representation of a large dataset for classification, regression, or generation. In this work, we describe a model that can flexibly perform both of these processes simultaneously, memorizing examples of rare items and learning consolidated representations of common ones. Additionally, we propose a biologically motivated implementation of two methods used in machine learning for learning on imbalanced data – loss reweighting and importance sampling [232].

Critically, unlike the traditional supervised setting, we consider autoassociative recall, an unsupervised learning setting where the class labels are not known to the learner a priori, and the loss weights or sampling probabilities must be inferred online during learning. Using a number of metrics extracted from the network to reweight or resample, we find that our model significantly outperforms the baseline unweighted performance, and in some cases approaches that of ground-truth reweighting using labels, permitting memorization and consolidation to occur in the same model. Moreover, this framework permits insight on the tradeoffs between these two processes in biological and artificial neural networks, suggesting potential signals to control the tradeoff, an optimal latent representation for these to co-exist, and considerations on the stability of learning.

We consider a framework inspired by the complementary learning systems theory [224, 225] in which a fast learner (e.g. the hippocampus) rapidly accumulates data, which it then "replays" [233] to a slow learner (e.g. the cortex) for consolidation and long-term storage. To model the cortex, we consider a modified version of the modern Hopfield network [234, 235, 70], trained with stochastic gradient descent. Although it may be possible to use a biologically plausible variant of backpropagation [236, 237], we compute gradients using the standard backpropagation through time (BPTT) algorithm.

We do not explicitly model the fast learner and simply assume that there exists a mechanism for storing a dataset in a buffer from which minibatches can be assembled for replay to the slow learner. Such a buffer might be implemented biologically by a fast memory system such as the classical Hopfield network with Hebbian learning rules and forgetting [230, 238], or a key-value memory network with three-factor learning rules and rapid overwriting [239]. As a further step towards biological plausibility, we also discuss a scenario that obviates the fast learner entirely and enables directly training the slow learner online without a buffer.

4.2 Related work

There are several areas of research with similar themes to our work, but with different goals and settings. For clarity, we outline them here and highlight the differences. The first is the broad area

of outlier/anomaly detection and rejection [240, 241, 242]. In our work, however, we emphasize storing and retrieving the rare items rather than rejecting them. This problem is, in turn, superficially related to training on imbalanced data. However, we consider the unsupervised learning scenario with unlabeled data, so standard techniques for learning such as loss reweighting or importance sampling [232] cannot be used out-of-the-box because we do not know which data points belong to the rare classes a priori.

Other work focuses on models that can rapidly acquire memories in parallel with a slow learning process [243]. However, fast learning means fast forgetting. We would like both common and rare items stored long-term by the slow learner, some consolidated and some memorized. Our work also differs from that on few-/one-shot learning [244]: we are modeling the process of memorization/consolidation, and not explicitly evaluating the network's ability to generalize from few examples (although this may nevertheless be the case for our proposed algorithm). Finally, a body of work investigates the double descent phenomenon [245] and the related case in which overparameterized neural networks successfully generalize despite having the capacity to memorize/overfit the entire training set [246, 95, 247]. Here, we focus on the more biologically relevant underparameterized scenario, which forces the network to compress/consolidate the training data while memorizing a critical subset.

We also highlight two pieces of work that have conceptual overlap with our approach. The first suggests two separate subnetworks for storing the regular and exceptional items, along with an anomaly detector to route examples to the appropriate subnetwork [248]. Another suggests the idea of dynamic loss functions, continuously re-weighting the loss for each class as training progresses to accelerate learning, although the authors only consider the supervised scenario with labeled data [249].

4.3 Methods

4.3.1 Data

We consider a dataset consisting of R "regular" items from a commonly occurring class, of which the learner sees many examples; and E "exceptions," which comprise a few examples from a rare class:

$$\mathcal{D} = \{x_1, \dots x_R, x_{R+1}, \dots, x_{R+E}\}, \text{ with } R \gg E \tag{4.1}$$

As a toy example, we use the MNIST dataset, where the "regular" subset consists of $R = 6000$ examples of zeroes, and the "exception" subset consists of $E = 3$ examples of ones. Critically, the data are not labeled, and the learner does not know in advance which items belong to the regular class or the exceptions. These must be inferred online over the course of learning. This furthermore allows the possibility of treating unusual examples of regular items as exceptions and storing them verbatim, which can play a role analogous to data pruning [250] or label memorization in learning long-tailed data distributions [251].

Importantly, the dataset size is much larger than the capacity of the network, so to successfully encode the dataset it is necessary to perform some sort of compression – for example storing only a subset of *exemplar* items, a set of *features* which can be assembled to recreate the dataset, a set of *prototypes* which are representative of the items, or a combination of these strategies.

4.3.2 Model

To model the (slow) learner, we consider a case of the modern Hopfield network (MHN) [70], consisting of a visible layer with activity given by the vector v and hidden layer h. The dynamics of this network are given by

$$\begin{aligned} \tau_h \frac{dh}{dt} &= -h + Wg(v) \\ \tau_v \frac{dv}{dt} &= -v + W^\top f(h) \end{aligned} \tag{4.2}$$

Extending the set of models described in [70], we introduce a new model in which the visible layer activity is restricted to lie on a unit sphere and the hidden layer has an attention mechanism, represented by the transfer functions $g(v) = \frac{v}{||v||}$ and $f(h) = \mathrm{softmax}(\beta h)$. Biologically these might be implemented by divisive normalization [252], lateral inhibition [253], or intermediate neurons [254]. The weight matrix $W \triangleq [\xi_1, \dots \xi_M]^\top$ is normalized such that $||\xi_\mu|| = 1$, ensuring that the softmax inverse-temperature parameter β is independent of the magnitude of the weights.

To run the network, we generate a perturbed input $\widetilde{x}_i$, corresponding to an item x_i from the dataset with a random 50% of the pixels set to zero. This is used as the initial state of the network $g(0)$. The unperturbed pixels are clamped to their initial values, and the rest evolve according to Eq. 4.2, discretized with $\tau_v = 1, dt = 0.05$. Assuming the dynamics of the hidden layer are much faster than those of the visible layer, we set $\tau_h = 0$, so that $h = Wg(v)$ for all t. We run the network until it reaches a fixed point, so $g(T + dt) = g(T) \triangleq g^*$, which is used as the network output.

Other autoassociative models can be used, such as an energy-based convolutional architecture [255], or undercomplete, sparse, or denoising autoencoders with or without convolutional layers. However, there are a number of advantages to using the MHN. First, the structure of the weight matrix allows us to directly look at and easily interpret the stored representations (Figures 4.2 and 4.7), differentiating them from the memories which correspond to the network's fixed points (Figure 4.1), as well as the hidden layer representations. Moreover, a rapidly growing transfer function such as the softmax hidden layer nonlinearity allows the network to have a supralinear storage capacity [234, 235, 256], and tuning its β parameter can place the network in different operating modes (see Section 4.4.1), storing features, prototypes, or intermediate representations. Finally, its relationship to the attention mechanism of the Transformer architecture [235] may enable us go gain theoretical insight into larger models, or aid us with engineering more robust applications.

4.3.3 Learning

We train the network on the autoassociative memory task with stochastic gradient descent [257]. On each iteration we draw a minibatch $\mathcal{B} \subset \mathcal{D}$ from the dataset, generate a perturbed input $\tilde{x}_b$ and

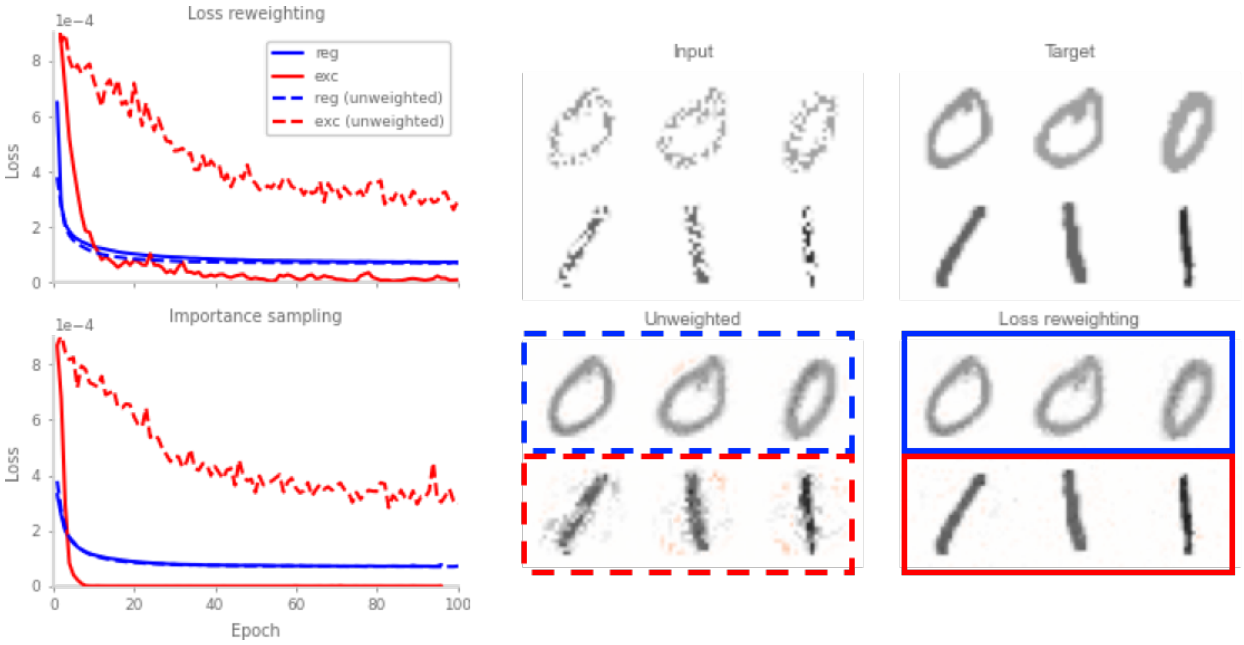


Figure 4.1: Both loss reweighting or importance sampling significantly improve memorizing exceptions without impairing performance on regular items. Left: comparison of weighted (solid) (top: loss reweighting, bottom: importance sampling) and unweighted (dashed) training, using $\beta = 5$. Right: sample outputs, trained without weighting/resampling and with loss reweighting.

compute the output g_b^* for every minibatch item x_b. We then calculate the loss $\mathcal{L}$ on each minibatch and update the parameters in proportion to the gradient of the loss $\nabla_\theta \mathcal{L}$ with respect to the network parameters θ. Classically, minibatches would be drawn uniformly without replacement so that every datapoint is seen exactly once per epoch. Furthermore, the loss would be the simple arithmetic mean (or sum) of individual losses ℓ_b for each item x_b in the minibatch. In the case of imbalanced data, a typical approach would be to compute a weighted loss for the minibatch, where the weights $\hat{\alpha}_b$ are taken inversely proportional to the relative frequency of the classes, assuming the class labels are available:

$$\mathcal{L} = \sum_{b=1}^{|\mathcal{B}|} \hat{\alpha}_b \, \ell(x_b, g_b^*) \tag{4.3}$$

Another common technique is to sample the minibatches *with* replacement according to a probability distribution inversely proportional to the relative frequency of the classes:

$$p(x_i \in \mathcal{B}) = \hat{\alpha}_i \tag{4.4}$$

In many situations for both biological and artificial agents, however, the class labels are not known a priori. In this work, we propose a number of metrics that can be used as dynamic signals s_i for reweighting or sampling, either individually or in combination. These are re-computed for each training iteration, and will vary over the course of training. To meaningfully compare the scores, we normalize them to the range $[0, 1]$ using the cumulative min/max of s_i seen so far during training:

$$\alpha_i = \frac{s_i - \min(s_{i'})}{\max(s_{i'}) - \min(s_{i'})} \tag{4.5}$$

The final loss weights or sampling probabilities are then given by

$$\hat{\alpha}_i = \frac{\exp(\gamma \alpha_i)}{\sum_{i'=1}^{|\mathcal{D}|} \exp(\gamma \alpha_{i'})} \tag{4.6}$$

where the inverse-temperature parameter γ controls the relative magnitude of the weights.

Notes on biological realism

So far, we have assumed that the entire dataset is fully available to the sampling/weighting mechanism (e.g. hippocampus). By definition, having the dataset in a buffer is required for importance sampling, since data are drawn with replacement. It is also necessary to continuously track α_i for every item, since they are all needed to re-compute the denominator in Eq. 4.6 at every iteration.

On the other hand, in the case of loss reweighting, instead of computing the denominator as the sum over the dataset, we can sum over the batch instead:

$$\hat{\alpha}_b = \frac{\exp(\gamma \alpha_b)}{\sum_{b'=1}^{|\mathcal{B}|} \exp(\gamma \alpha_{b'})} \tag{4.7}$$

In this case, the learner can receive batches from a stream of data, compute the weightings online and use them only for the current update. Although this introduces a dependence on the batch size and slightly different weightings, with a batch size of 100, we find that this approach also works

well on our dataset.

Alternatively, we can approximate the denominator in the expression for $\hat{\alpha}_i$ as $Z \approx \sum_{i'=1}^{|\mathcal{D}|} \exp(\gamma \alpha_{i'})$ as a running average:

$$Z \leftarrow (1-\lambda) Z + \lambda \sum_{b'=1}^{|\mathcal{B}|} \exp(\gamma \alpha_{b'}) \tag{4.8}$$

with $\lambda = \frac{B}{D}$. In this case, although the denominator is not exact, this can be done in a fully streaming fashion, receiving data one sample at a time and taking a gradient step for each sample.

4.4 Results

4.4.1 Feature-prototype transition

We begin by demonstrating the behavior of the modern Hopfield network in the imbalanced data autoassociative learning setting described above. First, training the network naively with no loss weighting or importance sampling leads to successfully learning the regular items, but failing to store the exceptions (Figure 4.1, dashed curves). (Although the loss on exceptions decreases, this is incidental due to overlapping pixels in MNIST 1's and 0's). On the other hand, a network trained with ground truth knowledge of whether an item is an exception – either upweighting or resampling exceptions – successfully learns both types of items, notably learning exceptions without degrading performance on regulars (Figure 4.1, solid curves). This holds true for either of the network's operating regimes (feature or prototype) which we describe next.

Although it has a minor effect on performance (Figure 4.6), the hidden layer inverse-temperature parameter β has a profound effect on the learned representations. Intuitively, as $\beta \to 0$, the hidden layer activity approaches a uniform distribution, with all units equally active. As $\beta \to \infty$, the population activity approaches a one-hot regime. Correspondingly, the weights transition from a *feature* regime where multiple rows ξ_μ are linearly combined to produce a given output (Figure 4.2, left) to a *prototype* regime where a single row is the output itself (Figure 4.2, right), with a smooth transition in between. This result is analogous to increasing the degree n of a polynomial hidden layer nonlinearity $f(h_\mu) = [h_\mu]_+^n$ as shown in [234], but notably generalizing to the $f(h) = \mathrm{softmax}(h)$

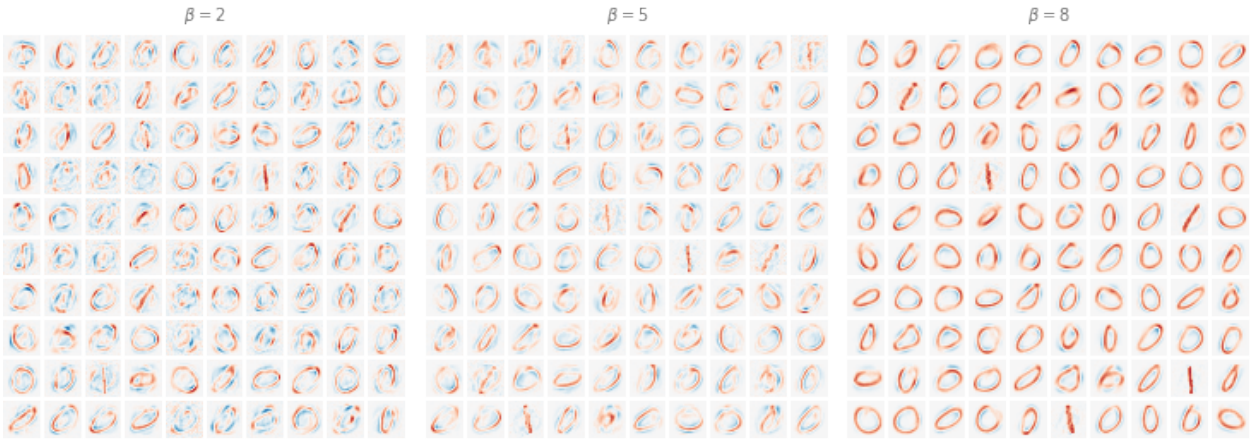


Figure 4.2: Increasing the inverse-temperature parameter β induces a feature-prototype transition. Learned weights $\{\xi_\mu\}$ in the feature (left), prototype (right) and intermediate (middle) regimes.

nonlinearity, as well as to training with imbalanced data.

Note that during training in the prototype regime, we linearly anneal the β parameter from $\beta = 5$ to $\beta = 8$ for the first 1000 iterations. This helps to avoid "dead units" which never get activated for any of the inputs in the training set and their corresponding weight representations never get updated.

4.4.2 Automatic memorization of exceptions

The primary contribution of this work is identifying signals that can be extracted directly from the network and used to automatically reweight or resample the training data. We identify possible candidate exception signals s_i by evaluating them as learning progresses for a network that is trained with with ground truth loss weighting, as in the previous section. We consider the following candidates:

$$s_i = \begin{cases} \mathbb{I}[x_i \in E] & \text{ground truth: control condition/evaluation metric} \\ \ell(x_i, g_i^*) & \text{loss per item} \\ -T & \text{(negative) time to converge to fixed point} \\ -H(f_i) & \text{(negative) entropy of hidden layer} \\ \max(f_i) & \text{max of hidden layer} \\ ||f_i|| & \text{norm of hidden layer} \end{cases} \tag{4.9}$$

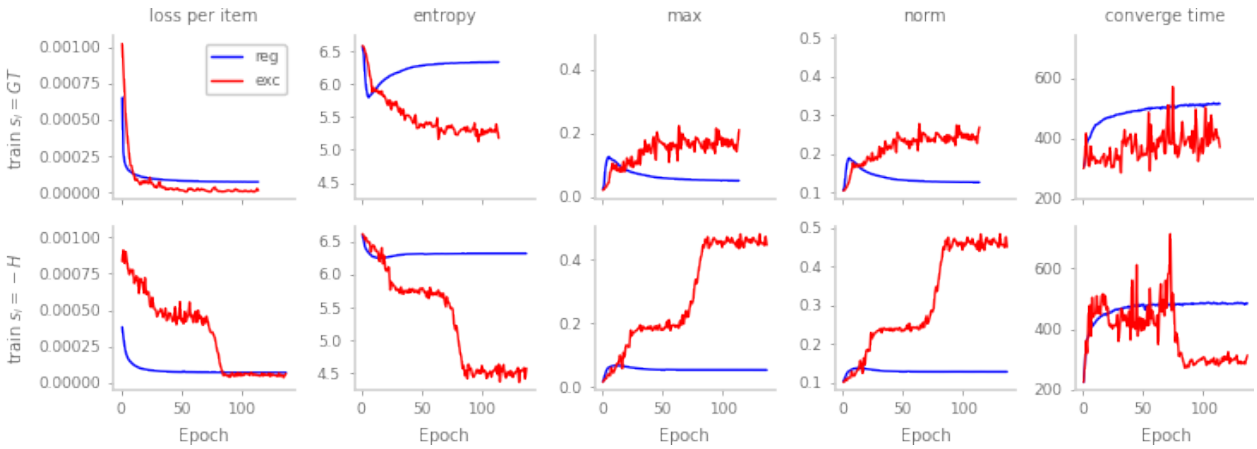


Figure 4.3: Signals extracted from the network can differentiate regular items from exceptions, and used to bootstrap learning for concurrent memorization and consolidation. Top: candidate signals for automatic loss weighting or importance sampling, measured as the network is trained using ground truth knowledge of whether each item is an exception. Bottom: the same signals, measured as the network is trained using negative entropy $s_i = -H(f)$ as the exception signal

Note that $\gamma = 0$ reduces to the unweighted case, and $s_i = \mathbb{I}[x_i \in E]$ with $\gamma = \ln \frac{R}{E}$ corresponds to the ground truth weighting in Eq. 4.3.

Several metrics can differentiate the regular items from exceptions when the network is trained with ground truth weighting (Figure 4.3, top), suggesting that it may be possible to use them to bootstrap the exception signal without prior knowledge of which items are exceptions. Note that some of the signals (entropy, converge time) have the opposite sign from the ground truth signal, so we use their negation as the signal s_i. Indeed, if we train the network using (negative) entropy as the exception signal, the regulars and exceptions are again differentiated, as shown in Figure 4.3 (bottom).

Training the network with either loss weighting or importance sampling, using each of the metrics identified above to compute the weights $\hat{\alpha}_i$, we compute the evaluation loss as follows:

$$\mathcal{L}_{\text{eval}} = \frac{1}{2}\left(\frac{1}{R}\sum_{i \in R} \ell(x_i, g_i^*) + \frac{1}{E}\sum_{i \in E} \ell(x_i, g_i^*)\right) \tag{4.10}$$

Note, this is equal to the ground-truth-weighted training loss (with $\gamma = \ln(\frac{R}{E})$). Nevertheless, this is an incomplete metric of successful learning, as it does not account for memorizing unusual

examples of the common class (i.e. correct treatment of some "regular" items as exceptions), nor does it evaluate generalization performance for unseen items from the common category.

The evaluation loss at the end of training in each case is shown in Figure 4.4, along with the optimal γ for each one, optimized though a linear hyperparameter search over $\gamma \in \{0, 1, 3, 5, 7.58, 10, 12, 15\}$ (note, $\gamma = \ln(\frac{R}{E}) \approx 7.58$ for this dataset). Overall, we see that using $\beta = 5$, corresponding to an intermediate regime between feature and prototype, has overall better performance. In the feature regime ($\beta = 2$), although the network is capable of successfully learning both regular and exceptional items using ground-truth weighting, the signals extracted from the network (with the notable exception of loss-per-item) are not sufficient to differentiate between the two types. In the prototype regime ($\beta = 10$), although the exception signals s_i are robust, the baseline unweighted performance (black lines) is worse, and so the weighted performance is correspondingly worse as well. This is either simply due to optimization difficulties (e.g. in Figure 4.2 we see "dead units" which do not correspond to a prototype and do not participate in the readout), or indicates that there may be an overall optimal level of sparsity [258] in the hidden representation for this task. Finally, counter-intuitively, we note that even when training with ground truth weighting, the optimal γ is not equal to $\ln(\frac{R}{E}) \approx 7.58$ (i.e. the effective weighting used in the evaluation loss).

4.4.3 Loss weighting vs. importance sampling

Although they have comparable performance (Figure 4.4) and generate similar memory representations $\{\xi_\mu\}$ (Figure 4.7), the learning trajectories generated by loss weighting and importance sampling are different. Evaluation loss (Eq. 4.10), computed over the entire dataset at the end of every epoch, decreases smoothly in both cases (Figure 4.5, top). Training loss, however, in the case of loss weighting is much more noisy (Figure 4.5, bottom), potentially leading to instability. This occurs because the exceptions are rarely seen by the learner, but when they are seen they generate a large gradient update due to their high weighting in the loss. On the other hand, importance sampling ensures that exceptions are seen in almost every batch, leading to many small updates, rather than a few large ones.

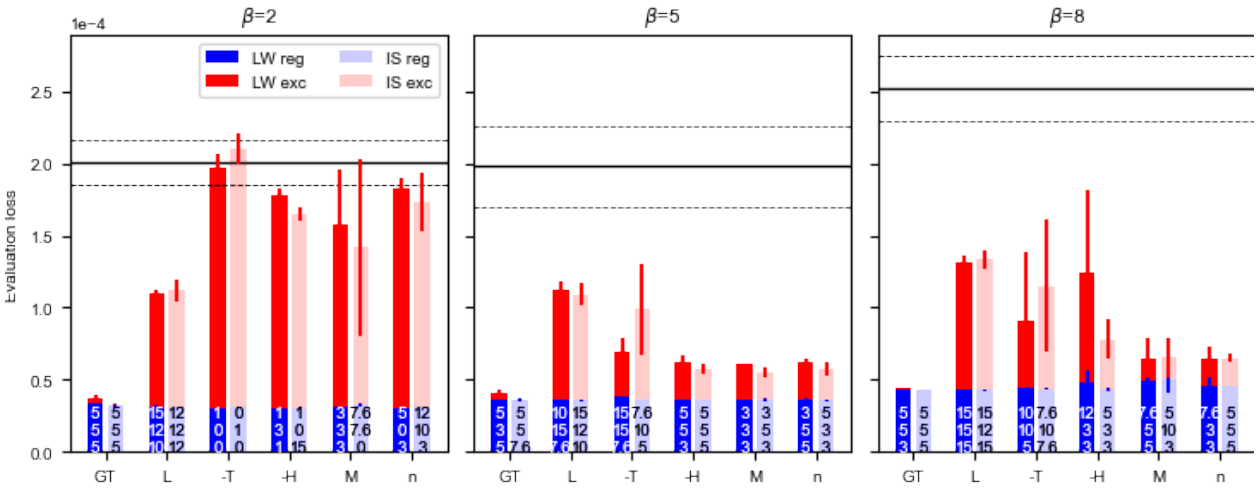


Figure 4.4: Evaluation loss after 150 epochs of training, split between loss on regulars (blue) and exceptions (red), trained with loss weighting (dark) or importance sampling (light) using exception signals s_i from Eq. 4.9. Numbers at the bottom of each bar indicate the optimal γ hyperparameter for that regime. Black lines indicate baseline performance with $\gamma = 0$ (no reweighting/resampling). Error bars and dashed lines indicate $\pm$ 1 standard deviation across $n = 3$ runs.

Therefore, from the perspective of learning, importance sampling is the preferred strategy, but as discussed in Section 4.3.3, it requires a buffer to store the entire training dataset (or to accumulate a subset of it) for sampling – a potential difficulty requiring additional machinery in the form of a fast learner for both biological and artificial systems. Furthermore, loss weighting is readily amenable to gradient-based meta-learning. On the other hand, importance sampling would require either evolutionary algorithms or techniques like the "reparameterization trick" [259] for optimizing through a random sampler.

4.5 Discussion

In this work we have proposed a biological framework for two related methods to concurrently memorize individual examples of rare categories and learn consolidated representations of common ones. Critically, we consider the unsupervised learning scenario where signals of exceptional items must be extracted from the network rather than given a priori as in the case of supervised learning with labelled imbalanced data. For this, we proposed a number of metrics that can be used as weights for loss reweighting or importance sampling and compared them across the feature and

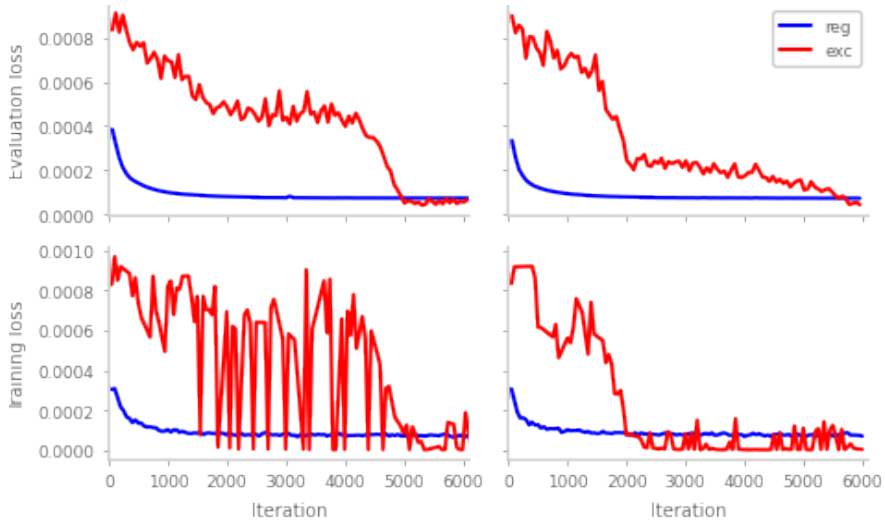


Figure 4.5: Training using importance sampling is significantly more stable than loss reweighting. Top: Evaluation loss trajectories with loss weighting (left) or importance sampling (right), using negative entropy as the exception signal ($\gamma = 5$). Bottom: Training loss for the same trials.

prototype network operating regimes.

Furthermore, our model has potential applications to machine learning and artificial intelligence. In recent years, it has been shown that large language models not only generate novel outputs but also output memorized sequences from their training corpus, raising significant privacy concerns [260]. Our network can be used as a simplified tractable model of this phenomenon to study this problem, particularly given its similarities to the attention mechanism of Transformers [235], enabling identification of memorized items in the weight matrix (Figure 4.2), and their removal or updating as necessary.

Finally, from the perspective of fairness in AI [261], our work suggests a potential technique to help mitigate some of the common problems arising from underrepresented classes, automatically balancing training on biased data by downweighting overrepresented classes and enhancing rare ones.

4.6 Supplementary figures

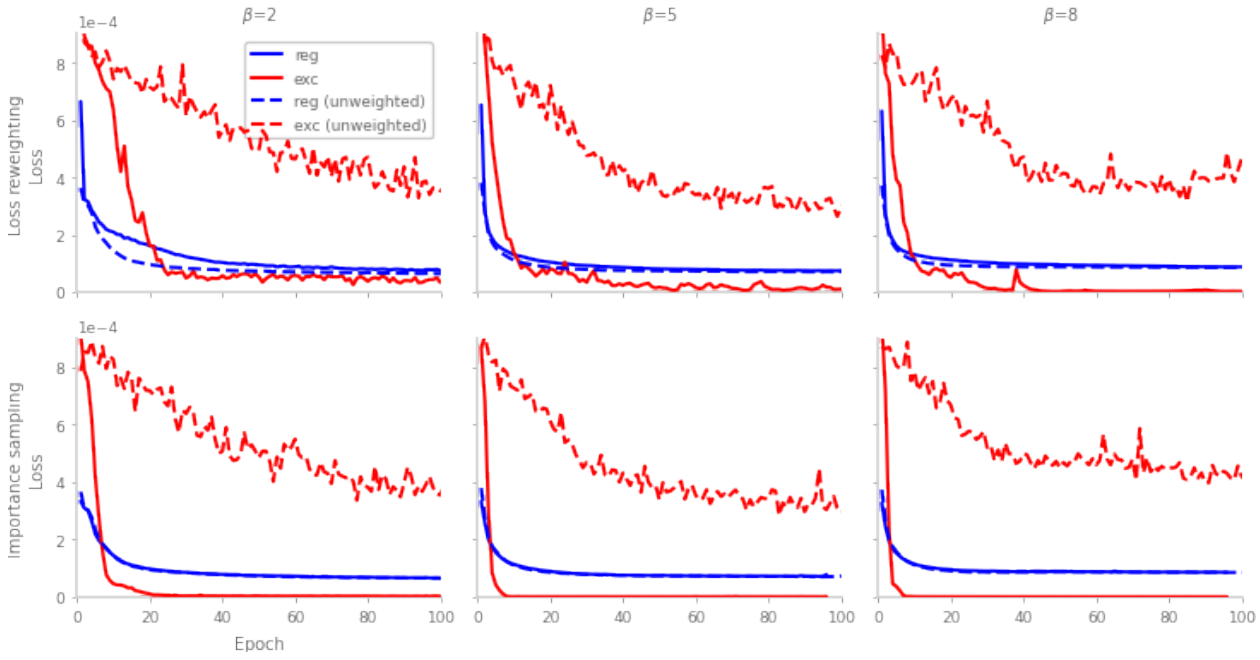


Figure 4.6: Same as Figure 4.1, shown in the feature ($\beta = 2$), prototype ($\beta = 8$) and intermediate ($\beta = 5$) regimes

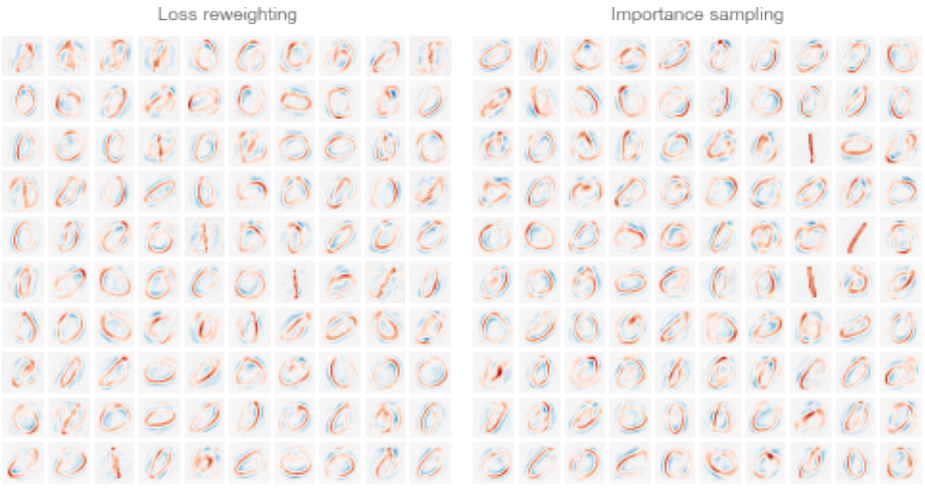


Figure 4.7: Learned weights $\{\xi_\mu\}$, trained with loss weighting or importance sampling, using negative entropy as the importance signal ($\gamma = 5$)

Chapter 5: Efficient recurrent backpropagation in modern Hopfield networks

The work presented in this chapter was done in collaboration with Dmitry Krotov. Thanks to Larry Abbott for continued guidance, mentorship, and support.

The Hopfield network is a long-standing model of memory, storing a predetermined set of memories according to a simple Hebbian rule, which are read out through recurrent attractor dynamics. More generally, however, the "memories" are not individual items or episodes to be stored verbatim, but are consolidated representations which enable generalization to new experiences. These representations must be statistically learned from data, for example with gradient descent.

The standard algorithm for computing the gradient in recurrent networks is backpropagation through time (BPTT), although it is considered to be biologically implausible as it requires neurons to store their entire history of activity to perform credit assignment [236]. An alternative known as recurrent backpropagation (RBP) was proposed by [262, 263], which relies on network activity converging to a fixed point, allowing computation of the gradient without knowledge of the trajectory. The calculation, however, involves a matrix inversion that is both biologically implausible and computationally expensive.

In this work, we propose an approximation of RBP for modern Hopfield networks (MHN), a generalization of the classical Hopfield network, which computes the gradient in a more efficient and biologically plausible manner than either BPTT or RBP. Although our approach is applicable to any Hopfield-like network, to demonstrate the algorithm we introduce a new special case of the MHN from [70]. In this case, we derive an approximation of the RBP gradient through a first-order Taylor expansion combined with a biologically motivated heuristic step. Finally, we demonstrate empirically that our approximation performs as well as the exact gradient calculation, and analyze its

time and memory complexity to show that it is more computationally efficient both asymptotically and in practice.

5.1 Introduction and related work

The classical Hopfield network [230] is a long-standing model of memory. It is a recurrent neural network with a symmetric weight matrix $W_{ij} = W_{ji}$ corresponding to synaptic strength, and discrete states $x_i^{(t)} \in \pm 1$ corresponding to active or silent neurons. For this system, we can define an energy function $E(\boldsymbol{x}) = -\boldsymbol{x}^\top \boldsymbol{W} \boldsymbol{x}$ which decreases monotonically according to the dynamics $x_i^{(t+1)} = \text{sign}\left(\sum_j W_{ij} x_j^{(t)}\right)$. Minima of E correspond to stable fixed-points of this dynamical system, and are controlled by the choice of weights $\boldsymbol{W}$.

The capacity (number of unique fixed points), has been be improved through alternative choices of E, most recently in "modern" Hopfield networks (MHNs) [234, 216, 235] which have been shown to be a powerful storage system for practical applications requiring high memory capacity [264]. These high-capacity energy functions, however, imply dynamics which require biologically implausible higher-order interactions among neurons, thus limiting their utility as neurobiological models. These issues have been partially addressed by introducing a layer of hidden neurons [70, 265] which can implement these energy functions with only pairwise neuronal interactions at the cost of more complex circuitry.

MHNs are furthermore limited by a lack of biologically plausible general learning rules. In the special case of storing a predetermined set of memories $\{\boldsymbol{\xi}^\mu\}_{\mu=1}^P$, classical Hopfield networks [230] let $W_{ij} = \sum_{\mu=1}^P \xi_i^\mu \xi_j^\mu$, corresponding to a simple Hebbian rule, which ensures that there exist fixed points at $\boldsymbol{x} = \boldsymbol{\xi}^\mu$. In MHNs with hidden units [70], memories can be stored in a biological way [239] with "three-factor" plasticity rules [266, 267]. More generally, however, the "memories" are not individual items or episodes to be memorized, but are consolidated representations which enable generalization to new experiences. For instance, in a classification task, the number of examples is typically several orders of magnitude larger than the number of classes, and the capacity of the network is not sufficient to store each one individually. Instead, the internal representations $\boldsymbol{\xi}^\mu$ must

be learned, for example with gradient descent [234].

The standard algorithm for computing the gradient in recurrent networks is backpropagation through time (BPTT). As stated, it is considered to be biologically implausible, as it requires neurons to store their entire history of activity to perform credit assignment [236]. There have been, however, many suggestions to create biological approximations and interpretations of this algorithm [268].

An alternative to BPTT was proposed in [262, 263], known as recurrent backpropagation (RBP). This algorithm relies on network activity converging to a fixed point, which allows computation of the gradient without knowledge of the trajectory. The calculation, however, involves a matrix inversion that is both biologically implausible and computationally expensive. In the original work, these issues were addressed by computing the inverse continuously through a secondary network [269]. More recent approaches approximate it via the Sherman-Morrison formula [270] or the Neumann series [271] in an efficient but non-biological manner.

In this work, we propose an implementation of RBP for modern Hopfield networks, which computes the gradient equivalent to BPTT but in a more computationally efficient and biologically plausible manner. Specifically, we consider a special case of a two-layer network [70] with a softmax nonlinearity in the hidden layer and derive a first-order approximation of the RBP gradient in the limit of a large inverse-temperature parameter. We demonstrate empirically that the heuristic approximation performs as well as the exact gradient calculation, and analyze its time and memory complexity in the two layer network to show that it is much more computationally efficient both asymptotically and in practice.

5.2 Results

5.2.1 Generalized modern Hopfield network

We begin by deriving the RPB gradient formula [262, 263] for the generalized formulation of the modern Hopfield network introduced in [255]. Consider a fully connected recurrent network with with a symmetric synaptic weight matrix $\boldsymbol{W}$. Let $\boldsymbol{x}$ be a vector where x_i is the input current to neuron i, and let $\boldsymbol{\phi} = \boldsymbol{\phi}(\boldsymbol{x})$ be the corresponding vector of neuronal firing rates. Note that unlike

typical recurrent networks, the nonlinearity is a general vector field and is not constrained to act element-wise on $\boldsymbol{x}$. The network dynamics evolve according to

$$\boldsymbol{\tau}\frac{d\boldsymbol{x}}{dt} = -\boldsymbol{x} + \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}) \tag{5.1}$$

where $\boldsymbol{\tau}$ is a diagonal matrix with τ_{ii} representing the membrane time constant of neuron i. If the Jacobian of $\boldsymbol{\phi}(\cdot)$ is positive semi-definite, then the network has a global energy function and therefore the dynamics are guaranteed to converge to a stable fixed point [255].

We are interested in training this network to minimize a loss function $\mathcal{L} = \mathcal{L}(\boldsymbol{\phi}, \widehat{\boldsymbol{\phi}})$ with respect to a network parameter θ, where $\widehat{\boldsymbol{\phi}}$ is the network target firing rate. The parameter θ may be one of the synaptic weights, or a parameter of the nonlinearity $\boldsymbol{\phi}(\cdot)$. We suppose that the network activity has reached a fixed point, i.e. $\frac{d\boldsymbol{x}}{dt} = 0$, which implies that $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x})$. Taking the derivative of the loss with respect to θ, we have

$$\frac{d\mathcal{L}}{d\theta} = (\nabla_{\phi}\mathcal{L})^{\top}\frac{d\boldsymbol{\phi}}{d\theta} \tag{5.2}$$

Next, we must compute $\frac{d\boldsymbol{\phi}}{d\theta}$, as

$$\frac{d\boldsymbol{\phi}}{d\theta} = \frac{\partial\boldsymbol{\phi}}{\partial\theta} + \boldsymbol{J}\frac{d\boldsymbol{x}}{d\theta} \tag{5.3}$$

$$= \frac{\partial\boldsymbol{\phi}}{\partial\theta} + \boldsymbol{J}\frac{d(\boldsymbol{W}\boldsymbol{\phi})}{d\theta} \tag{5.4}$$

$$= \frac{\partial\boldsymbol{\phi}}{\partial\theta} + \boldsymbol{J}\left(\frac{d\boldsymbol{W}}{d\theta}\boldsymbol{\phi} + \boldsymbol{W}\frac{d\boldsymbol{\phi}}{d\theta}\right) \tag{5.5}$$

where $J_{ij} = \frac{\partial\phi_i}{\partial x_j}$. Note that in Eq. 5.4 we have imposed the fixed point condition. Rearranging,

$$(\boldsymbol{I} - \boldsymbol{J}\boldsymbol{W})\frac{d\boldsymbol{\phi}}{d\theta} = \frac{\partial\boldsymbol{\phi}}{\partial\theta} + \boldsymbol{J}\frac{d\boldsymbol{W}}{d\theta}\boldsymbol{\phi} \tag{5.6}$$

Finally, solving for $\frac{d\boldsymbol{\phi}}{d\theta}$, we can write the gradient as

$$\frac{d\mathcal{L}}{d\theta} = \left(\nabla_{\boldsymbol{\phi}}\mathcal{L}\right)^{\top}\left(\boldsymbol{I} - \boldsymbol{J}\boldsymbol{W}\right)^{-1}\left(\frac{\partial\boldsymbol{\phi}}{\partial\theta} + \boldsymbol{J}\frac{d\boldsymbol{W}}{d\theta}\boldsymbol{\phi}\right) \tag{5.7}$$

Note that this is equivalent to the gradient computed by backpropagation through time – the standard algorithm for training recurrent neural networks. However, when the network activity is at a fixed point, the gradient is independent of the network's trajectory, thus avoiding the issue of credit assignment through time [236], and adding a degree of biological plausibility. However, this computation introduces a matrix inversion operation. We address this issue in the following section.

5.2.2 Two layer network: spherical memory with attention

As a concrete example, we consider a special case of the network introduced in the previous section. Letting

$$\boldsymbol{\phi} = \begin{bmatrix}\boldsymbol{g}\\ \boldsymbol{f}\end{bmatrix}, \boldsymbol{W} = \begin{bmatrix}0 & \boldsymbol{\xi}^{\top}\\ \boldsymbol{\xi} & 0\end{bmatrix}, \boldsymbol{J} = \begin{bmatrix}\boldsymbol{J}_{\boldsymbol{g}} & 0\\ 0 & \boldsymbol{J}_{\boldsymbol{f}}\end{bmatrix} \tag{5.8}$$

the generalized network in Eq. 5.1 reduces to the two-layer network introduced in [70], with a layer of feature neurons projecting to a hidden layer through a synaptic weight matrix $\boldsymbol{\xi}$, and hidden neurons projecting back to the feature neurons through symmetric weights $\boldsymbol{\xi}^{\top}$. The feature neuron and hidden neuron input currents are given by the vectors $\boldsymbol{v}$ and $\boldsymbol{h}$ and corresponding firing rates $\boldsymbol{g}(\boldsymbol{v})$ and $\boldsymbol{f}(\boldsymbol{h})$. The dynamics of this network are given by

$$\tau_h\frac{d\boldsymbol{h}}{dt} = -\boldsymbol{h} + \boldsymbol{\xi}\boldsymbol{g}(\boldsymbol{v}) \tag{5.9}$$

$$\tau_v\frac{d\boldsymbol{v}}{dt} = -\boldsymbol{v} + \boldsymbol{\xi}^{\top}\boldsymbol{f}(\boldsymbol{h}) \tag{5.10}$$

Assuming the loss only depends on the feature neurons, $\mathcal{L} = \mathcal{L}(\boldsymbol{g}, \hat{\boldsymbol{g}})$, the gradient with respect to the weights reduces to

$$\frac{d\mathcal{L}}{d\boldsymbol{\xi}} = \boldsymbol{f}\boldsymbol{a}^\top + \boldsymbol{J}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{a}\boldsymbol{g}^\top \tag{5.11}$$

where

$$\boldsymbol{a}^\top = (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{A}^{-1}\boldsymbol{J}_{\boldsymbol{g}} \tag{5.12}$$

$$\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{J}_{\boldsymbol{g}}\boldsymbol{\xi}^\top \boldsymbol{J}_{\boldsymbol{f}}\boldsymbol{\xi} \tag{5.13}$$

See Section 5.3.1 for a full derivation. Specifically, extending the set of models described in [70], we introduce a new model, where the feature layer activity is restricted to lie on a unit sphere and the hidden layer has an attention mechanism represented by the transfer functions

$$\boldsymbol{g}(\boldsymbol{v}) = \frac{\boldsymbol{v}}{||\boldsymbol{v}||}, \text{ and } \boldsymbol{f}(\boldsymbol{h}) = \mathrm{softmax}(\beta \boldsymbol{h}) \tag{5.14}$$

The energy function for this network reduces from the general formula in [42] and is given in Section 5.3.2. Biologically these transfer functions can be implemented, for example, through divisive normalization [252] and recurrent inhibition [254], respectively. Their Jacobians functions are given by

$$\boldsymbol{J}_{\boldsymbol{g}} = \frac{\boldsymbol{I} - \boldsymbol{g}\boldsymbol{g}^\top}{||\boldsymbol{v}||}, \text{ and } \boldsymbol{J}_{\boldsymbol{f}} = \beta\big(\mathrm{diag}(\boldsymbol{f}) - \boldsymbol{f}\boldsymbol{f}^\top\big) \tag{5.15}$$

Finally, as with the weights $\boldsymbol{\xi}$, we can train β , with the gradient given by (Section 5.3.1)

$$\frac{d\mathcal{L}}{d\beta} = \boldsymbol{a}^\top \boldsymbol{\xi}^\top \left(\frac{1}{\beta}\boldsymbol{J}_{\boldsymbol{f}}\boldsymbol{h}\right) \tag{5.16}$$

5.2.3 Speeding up gradient computation

Although in general it takes $O(M^3)$ time to invert the $M \times M$ matrix $\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{J_g}\boldsymbol{\xi}^\top \boldsymbol{J_f}\boldsymbol{\xi}$, we consider a further special case, letting $\beta \to \infty$. In this limit, we can compute the first-order approximation of the gradient with respect to the parameter $\boldsymbol{\varepsilon}$, the μ^{th} entry of which is given by

$$\varepsilon_\mu = e^{\beta(h_\mu - h_{\mu^*})} \tag{5.17}$$

where $\mu^* = \text{argmax}(\boldsymbol{h})$. The exact and approximate gradients are then given by

$$\frac{d\mathcal{L}}{d\boldsymbol{\xi}} = \boldsymbol{f}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{A}^{-1}\boldsymbol{J_g} + \boldsymbol{J_f}^\top \boldsymbol{\xi}\boldsymbol{J_g}^\top (\boldsymbol{A}^{-1})^\top (\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top \tag{5.18}$$

$$\approx \hat{\boldsymbol{e}}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top (\boldsymbol{I} + \boldsymbol{J_g}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi})\boldsymbol{J_g} + \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{J_g}^\top (\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top \tag{5.19}$$

where

$$\widetilde{\boldsymbol{J}}_{\boldsymbol{f}} = \beta \left[\text{diag}(\boldsymbol{\varepsilon}) - \boldsymbol{\varepsilon}\hat{\boldsymbol{e}}^\top - \hat{\boldsymbol{e}}\boldsymbol{\varepsilon}^\top + \left(\sum_\nu \varepsilon_\nu \right) \hat{\boldsymbol{e}}\hat{\boldsymbol{e}}^\top \right] \tag{5.20}$$

is the first-order approximation of $\boldsymbol{J_f}$ and $\hat{\boldsymbol{e}}$ is the leading-order of $\boldsymbol{f}$ (i.e. a one-hot vector with a 1 at entry μ^* and 0 otherwise). We refer to this algorithm as "RBP-1." See Section 5.3.3 for full derivation.

Next, as a heuristic, we can re-introduce (arbitrary) higher-order corrections by using the exact hidden layer activity $\boldsymbol{f}$ instead of its limiting value $\hat{\boldsymbol{e}}$, and the full Jacobian $\boldsymbol{J_f}$ instead of $\widetilde{\boldsymbol{J}}_{\boldsymbol{f}}$. Importantly, this heuristic ("RBP-1H") is more biological than the true first-order approximation since computing it uses the hidden layer activity directly, rather than a surrogate.

$$\frac{d\mathcal{L}}{d\boldsymbol{\xi}} \approx \boldsymbol{f}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top (\boldsymbol{I} + \boldsymbol{J_g}\boldsymbol{\xi}^\top \boldsymbol{J_f}\boldsymbol{\xi})\boldsymbol{J_g} + \boldsymbol{J_f}^\top \boldsymbol{\xi}\boldsymbol{J_g}^\top (\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top \tag{5.21}$$

This formulation makes the relationship to Hebbian learning more evident – the leading term of

the gradient, $\boldsymbol{f}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top$ is a local plasticity rule, where the pre-synaptic term is an error term (e.g. in the case of MSE loss $\nabla_{\boldsymbol{g}}\mathcal{L} = \boldsymbol{g} - \hat{\boldsymbol{g}}$) and the post-synaptic term is the hidden layer activation $\boldsymbol{f}$. The other terms are cubic or quartic in $\boldsymbol{g}$ and provide higher-order corrections to this simple update.

5.2.4 Empirical validation

To validate the RBP gradient computation, as well as the first-order approximation and the heuristic, we train this network on the MNIST dataset in three regimes:

1. *Memorization*. The dataset is small enough ($D = 25$ datapoints) relative to the network size ($N = 25$ hidden units) such that it can memorize the entire dataset verbatim, and theoretically have zero loss.

2. *Small network learning*. The dataset is very large ($D = 50,000$) relative to the network size ($N = 25$).

3. *Large network learning*. The dataset is large ($D = 50,000$), but the network size ($N = 500$) is sufficient to reliably capture the dataset statistics.

We train the networks on an autoassociative recall task with gradient descent, where we randomly set 50% of the input pixels to zero and use the original image as the target. We initialize the network's feature neurons with the perturbed input image and let the dynamics evolve according to Eq. 5.10 until they converge to a fixed point. Convergence is guaranteed due to the existence of an energy function for these dynamics (Section 5.3.2). Using the feature layer's state as the output, we compute the mean-squared error (MSE) loss between the output and the target, and compute the gradient using one of the three methods described here, as well as BPTT for comparison.

In all three regimes, the performance of RBP is identical to BPTT, by definition. Using the first-order approximation (RBP-1) results in a lower performance, indicating that the approximation breaks down for smaller values of β. Most importantly, the heuristic (RBP-1H) performs as well as the exact gradient computation, and may even converge to a minimum faster than RBP or BPTT in the memory regime (Figure 5.1). Example outputs are shown in Figure 5.2.

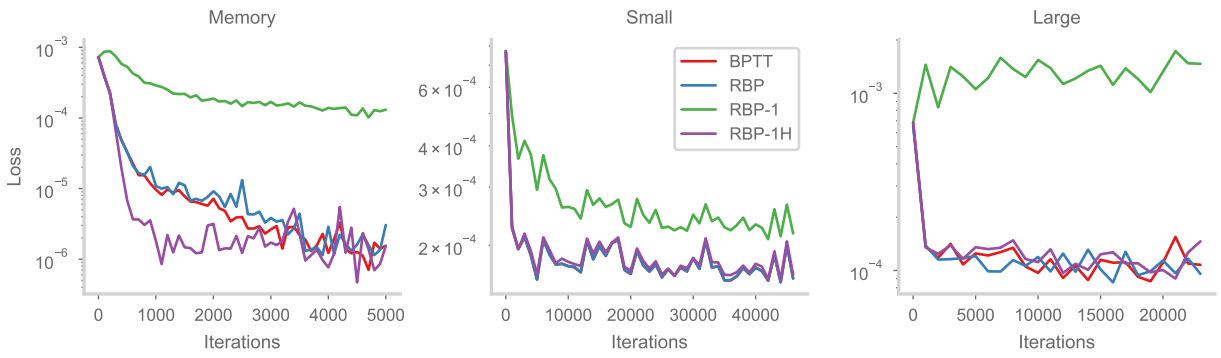


Figure 5.1: Training and test loss of spherical memory with attention trained on MNIST with three different memory loads.

Figure 5.2: Example outputs for the three different memory loads (columns) and three training algorithms (rows). Input examples were included in each network's training set.

5.2.5 Time and memory complexity

To compare the efficiency of BPTT with our proposed training algorithms, we consider training the two-layer network with M feature neurons and N hidden neurons, and suppose that the network activity reaches a fixed point in T timesteps.The RBP-1 or RBP-1H algorithms are faster by a factor of two, and use a factor of $O(T)$ less memory than BPTT. Using RBP is only advantageous for large T and N, as is the case in our examples. Results are summarized in Table 5.1. These results also hold empirically, as shown in Figure 5.3. The derivation is described in Section 5.3.4

	BPTT	RBP	RBP-1/-1H
time	$O(TMN)$	$O(TMN + M^3)$	$O(TMN)$
ratio	1	$O(\frac{TN}{M^2})$	≈ 2
memory	$O(T(M+N))$	$O(M^3 + N)$	$O(M+N)$
ratio	1	$O(\frac{TM+TN}{M^3+N})$	$O(T)$

Table 5.1: Time and memory complexity for one step of the gradient update, as well as the ratio compared to BPTT

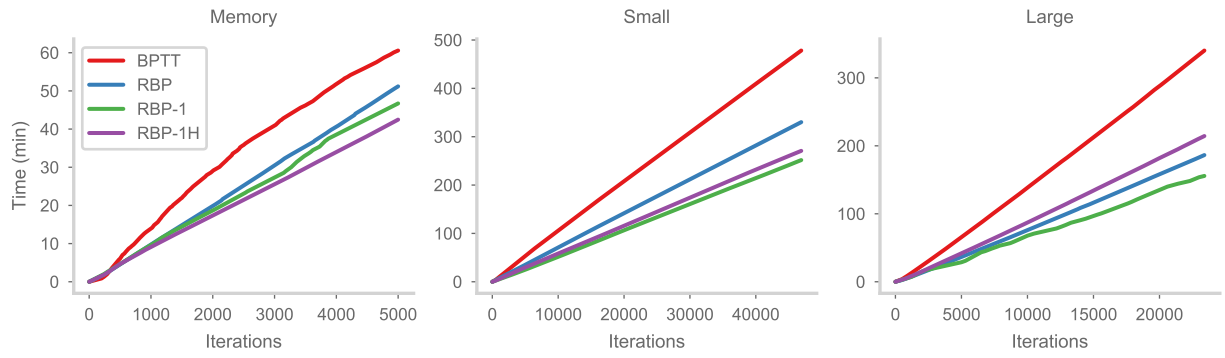


Figure 5.3: Time to complete a fixed number of training iterations for BPTT, RBP, and its two approximations. Differences in time are most profound for larger networks and larger batch sizes.

5.3 Supplementary Materials

5.3.1 Two-layer network gradient derivation

Substituting the values in Eq. 5.8 into Eq. 5.7, we get

$$\frac{d\mathcal{L}}{d\theta} = (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \left(\boldsymbol{I} - \boldsymbol{J_g}\boldsymbol{\xi}^\top \boldsymbol{J_f}\boldsymbol{\xi}\right)^{-1} \left[\boldsymbol{J_g}\left(\boldsymbol{\xi}^\top \left[\boldsymbol{J_f}\frac{d\boldsymbol{\xi}}{d\theta}\boldsymbol{g} + \frac{\partial \boldsymbol{f}}{\partial\theta}\right] + \frac{d\boldsymbol{\xi}^\top}{d\theta}\boldsymbol{f}\right) + \frac{\partial \boldsymbol{g}}{\partial\theta}\right] \tag{5.22}$$

Next, considering one of the synaptic weights $\xi_{\mu i}$ for the parameter θ, since $\frac{\partial g}{\partial \xi_{\mu i}} = 0$ and $\frac{\partial f}{\partial \xi_{\mu i}} = 0$

$$\frac{d\mathcal{L}}{d\xi_{\mu i}} = (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{A}^{-1}\left[\boldsymbol{J_g}\left(\boldsymbol{\xi}^\top \boldsymbol{J_f}\frac{d\boldsymbol{\xi}}{d\xi_{\mu i}}\boldsymbol{g} + \frac{d\boldsymbol{\xi}^\top}{d\xi_{\mu i}}\boldsymbol{f}\right)\right] \tag{5.23}$$

$$= (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{A}^{-1}\boldsymbol{J_g}\boldsymbol{\xi}^\top \boldsymbol{J_f}\frac{d\boldsymbol{\xi}}{d\xi_{\mu i}}\boldsymbol{g} + (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{A}^{-1}\boldsymbol{J_g}\frac{d\boldsymbol{\xi}^\top}{d\xi_{\mu i}}\boldsymbol{f} \tag{5.24}$$

$$= \boldsymbol{c}^\top \frac{d\boldsymbol{\xi}}{d\xi_{\mu i}}\boldsymbol{g} + \boldsymbol{d}^\top \frac{d\boldsymbol{\xi}^\top}{d\xi_{\mu i}}\boldsymbol{f} \tag{5.25}$$

where we have introduced convenience variables $\boldsymbol{c}$ and $\boldsymbol{d}$. In matrix notation,

$$\frac{d\mathcal{L}}{d\boldsymbol{\xi}} = \boldsymbol{c}\boldsymbol{g}^\top + \boldsymbol{f}\boldsymbol{d}^\top \tag{5.26}$$

$$= \boldsymbol{J_f}^\top \boldsymbol{\xi}\boldsymbol{J_g}^\top (\boldsymbol{A}^\top)^{-1}(\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top + \boldsymbol{f}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{A}^{-1}\boldsymbol{J_g} \tag{5.27}$$

$$= \boldsymbol{J_f}^\top \boldsymbol{\xi}\boldsymbol{a}\boldsymbol{g}^\top + \boldsymbol{f}\boldsymbol{a}^\top \tag{5.28}$$

If $\boldsymbol{f}(\boldsymbol{h}) = \mathrm{softmax}(\beta\boldsymbol{h})$, we can also compute the gradient with respect to β, using $\frac{d\boldsymbol{\xi}}{d\beta} = 0$, $\frac{d\boldsymbol{\xi}^\top}{d\beta} = 0$, $\frac{\partial g}{\partial \beta} = 0$:

$$\frac{d\mathcal{L}}{d\beta} = (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \left(\boldsymbol{I} - \boldsymbol{J_g}\boldsymbol{\xi}^\top \boldsymbol{J_f}\boldsymbol{\xi}\right)^{-1} \boldsymbol{J_g}\boldsymbol{\xi}^\top \frac{\partial \boldsymbol{f}}{\partial\beta} \tag{5.29}$$

$$= \boldsymbol{a}^\top \boldsymbol{\xi}^\top \left(\frac{1}{\beta}\boldsymbol{J_f}\boldsymbol{h}\right) \tag{5.30}$$

5.3.2 Energy function: Spherical memory with attention

Assuming that the dynamics of the hidden units are fast relative to those of the feature neurons, we let $\tau_h \to 0$, so $\boldsymbol{h}^{(t)} = \boldsymbol{\xi}\boldsymbol{g}^{(t)}$. In this case, the energy function is given by

$$E(t) = -\frac{1}{\beta}\log\left(\sum_{\mu}\exp\left(\beta\sum_{i}\xi_{\mu i}\frac{v_i}{\sqrt{\sum_j v_j^2}}\right)\right) \tag{5.31}$$

5.3.3 Two-layer network gradient approximation

In deriving the first-order approximation of the gradient $\frac{d\mathcal{L}}{d\boldsymbol{\xi}}$, we consider the two layer network with the hidden layer nonlinearity $\boldsymbol{f}(\boldsymbol{h}) = \mathrm{softmax}(\beta\boldsymbol{h})$, and an arbitrary feature layer nonlinearity $\boldsymbol{g}(\boldsymbol{v})$.

First, we compute the first-order approximation of $\boldsymbol{f}(\cdot)$ with respect to the parameter $\boldsymbol{\varepsilon}$, the μ^{th} entry of which is given by

$$\varepsilon_\mu = e^{\beta(h_\mu - h_{\mu^*})} \tag{5.32}$$

where $\mu^* = \mathrm{argmax}(\boldsymbol{h})$. In the limit we have

$$\lim_{\beta\to\infty}\boldsymbol{\varepsilon} = \hat{\boldsymbol{e}}$$

which is a one-hot vector with a 1 at entry μ^* and 0 otherwise. Considering $\boldsymbol{f}$ as a function of $\boldsymbol{\varepsilon}$, where $f_\mu = \frac{\varepsilon_\mu}{\sum_\nu \varepsilon_\nu}$, we can expand $\boldsymbol{f}$ about $\boldsymbol{\varepsilon} = \hat{\boldsymbol{e}}$

$$\boldsymbol{f}(\boldsymbol{\varepsilon}) \approx \boldsymbol{f}(\hat{\boldsymbol{e}}) + \frac{\partial \boldsymbol{f}(\hat{\boldsymbol{e}})}{\partial \boldsymbol{\varepsilon}}(\boldsymbol{\varepsilon} - \hat{\boldsymbol{e}}) \tag{5.33}$$

To get the leading order term, we have

$$\boldsymbol{f}(\hat{\boldsymbol{e}}) = \hat{\boldsymbol{e}} \tag{5.34}$$

Next, entry (μ, ν) of the first-order coefficient matrix $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\varepsilon}}$ is given by

$$\frac{\partial f_\mu}{\partial \varepsilon_\nu} = \frac{\partial}{\partial \varepsilon_\nu}\Big(\frac{\varepsilon_\mu}{\sum_\sigma \varepsilon_\sigma}\Big) \tag{5.35}$$

$$= \varepsilon_\mu \frac{\partial}{\partial \varepsilon_\nu}\Big(\frac{1}{\sum_\sigma \varepsilon_\sigma}\Big) + \frac{1}{\sum_\sigma \varepsilon_\sigma}\frac{\partial \varepsilon_\mu}{\partial \varepsilon_\nu} \tag{5.36}$$

$$= -\frac{\varepsilon_\mu}{\left(\sum_\sigma \varepsilon_\sigma\right)^2} + \frac{\delta_{\mu\nu}}{\sum_\sigma \varepsilon_\sigma} \tag{5.37}$$

$$\tag{5.38}$$

Evaluating it at $\hat{\boldsymbol{e}}$, we have

$$\frac{\partial f_\mu(\hat{\boldsymbol{e}})}{\partial \varepsilon_\nu} = -\delta_{\mu\mu^*} + \delta_{\mu\nu} \tag{5.39}$$

In matrix form,

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\varepsilon}} = \boldsymbol{I} - \hat{\boldsymbol{E}} \tag{5.40}$$

where $\hat{\boldsymbol{E}}$ is a square matrix with 1's in row μ^* and 0 otherwise. Thus, the first order approximation of $\boldsymbol{f}$ is given by

$$\boldsymbol{f} \approx \hat{\boldsymbol{e}} + (\boldsymbol{I} - \hat{\boldsymbol{E}})(\boldsymbol{\varepsilon} - \hat{\boldsymbol{e}}) \tag{5.41}$$

$$= \hat{\boldsymbol{e}} + (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon} \tag{5.42}$$

$$= (1 - \sum_\nu \varepsilon_\nu)\hat{\boldsymbol{e}} + \boldsymbol{\varepsilon} \tag{5.43}$$

$$\triangleq \tilde{\boldsymbol{f}} \tag{5.44}$$

which we define as $\tilde{\boldsymbol{f}}$. Using this, we calculate the first-order approximation of the Jacobian $\boldsymbol{J_f}$:

$$\boldsymbol{J_f} \approx \beta\big(\mathrm{diag}(\tilde{\boldsymbol{f}}) - \tilde{\boldsymbol{f}}\tilde{\boldsymbol{f}}^\top\big) \tag{5.45}$$

$$\approx \beta\big[\mathrm{diag}(\hat{\boldsymbol{e}} + (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}) - \hat{\boldsymbol{e}}\hat{\boldsymbol{e}}^\top - (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}\hat{\boldsymbol{e}}^\top - \hat{\boldsymbol{e}}\boldsymbol{\varepsilon}^\top(\boldsymbol{I} - \hat{\boldsymbol{E}})^\top\big] \tag{5.46}$$

$$= \beta\big[\mathrm{diag}((\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}) - (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}\hat{\boldsymbol{e}}^\top - \hat{\boldsymbol{e}}\boldsymbol{\varepsilon}^\top(\boldsymbol{I} - \hat{\boldsymbol{E}})^\top\big] \tag{5.47}$$

$$= \beta\left[\mathrm{diag}(\boldsymbol{\varepsilon}) - \boldsymbol{\varepsilon}\hat{\boldsymbol{e}}^\top - \hat{\boldsymbol{e}}\boldsymbol{\varepsilon}^\top + \left(\sum_\nu \varepsilon_\nu\right)\hat{\boldsymbol{e}}\hat{\boldsymbol{e}}^\top\right] \tag{5.48}$$

$$\triangleq \widetilde{\boldsymbol{J}}_{\boldsymbol{f}} \tag{5.49}$$

where we used $\hat{\boldsymbol{e}}\hat{\boldsymbol{e}}^\top = \mathrm{diag}(\hat{\boldsymbol{e}})$ and dropped the $O(\boldsymbol{\varepsilon}^2)$ term of $\tilde{\boldsymbol{f}}\tilde{\boldsymbol{f}}^\top$ as follows:

$$\tilde{\boldsymbol{f}}\tilde{\boldsymbol{f}}^\top = \big[\hat{\boldsymbol{e}} + (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}\big]\big[\hat{\boldsymbol{e}} + (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}\big]^\top \tag{5.50}$$

$$= \hat{\boldsymbol{e}}\hat{\boldsymbol{e}}^\top + (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}\hat{\boldsymbol{e}}^\top + \hat{\boldsymbol{e}}\boldsymbol{\varepsilon}^\top(\boldsymbol{I} - \hat{\boldsymbol{E}})^\top + (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top(\boldsymbol{I} - \hat{\boldsymbol{E}})^\top \tag{5.51}$$

$$\approx \hat{\boldsymbol{e}}\hat{\boldsymbol{e}}^\top + (\boldsymbol{I} - \hat{\boldsymbol{E}})\boldsymbol{\varepsilon}\hat{\boldsymbol{e}}^\top + \hat{\boldsymbol{e}}\boldsymbol{\varepsilon}^\top(\boldsymbol{I} - \hat{\boldsymbol{E}})^\top \tag{5.52}$$

Next, we compute the first-order approximation of the inverse of $\boldsymbol{A}$. In general, for any invertible matrix $\boldsymbol{I} - \boldsymbol{M}$, we have the Neumann series,

$$(\boldsymbol{I} - \boldsymbol{M})^{-1} = \sum_{n=0}^{\infty} \boldsymbol{M}^n \tag{5.53}$$

since (analogously to the geometric series)

$$(\boldsymbol{I} - \boldsymbol{M})^{-1} = \boldsymbol{I} + \boldsymbol{M} + \boldsymbol{M}^2 + \cdots \tag{5.54}$$

$$(\boldsymbol{I} - \boldsymbol{M})(\boldsymbol{I} - \boldsymbol{M})^{-1} = (\boldsymbol{I} - \boldsymbol{M})(\boldsymbol{I} + \boldsymbol{M} + \boldsymbol{M}^2 + \cdots) \tag{5.55}$$

$$\boldsymbol{I} = (\boldsymbol{I} + \boldsymbol{M} + \boldsymbol{M}^2 + \cdots) - \boldsymbol{M}(\boldsymbol{I} + \boldsymbol{M} + \boldsymbol{M}^2 + \cdots) \tag{5.56}$$

$$= (\boldsymbol{I} + \boldsymbol{M} + \boldsymbol{M}^2 + \cdots) - (\boldsymbol{M} + \boldsymbol{M}^2 + \boldsymbol{M}^3 + \cdots) \tag{5.57}$$

$$= \boldsymbol{I} \tag{5.58}$$

Thus, we can approximate the inverse of $\boldsymbol{A}$ to first order as

$$\boldsymbol{A}^{-1} = (\boldsymbol{I} - \boldsymbol{J_g}\boldsymbol{\xi}^\top \boldsymbol{J_f}\boldsymbol{\xi})^{-1} \tag{5.59}$$

$$\approx \boldsymbol{I} + \boldsymbol{J_g}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi} \tag{5.60}$$

Finally, we compute the first-order approximation of the gradient. Starting with the exact expression,

$$\frac{d\mathcal{L}}{d\boldsymbol{\xi}} = \boldsymbol{f}\boldsymbol{a}^\top + \boldsymbol{J}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{a}\boldsymbol{g}^\top \tag{5.61}$$

where

$$\boldsymbol{f}\boldsymbol{a}^\top = \boldsymbol{f}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{A}^{-1}\boldsymbol{J_g} \tag{5.62}$$

$$\approx \tilde{\boldsymbol{f}}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top (\boldsymbol{I} + \boldsymbol{J_g}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi})\boldsymbol{J_g} \text{ (use first-order approx of } \boldsymbol{f}, \boldsymbol{J_f}, \boldsymbol{A}) \tag{5.63}$$

$$= \tilde{\boldsymbol{f}}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{J_g} + \tilde{\boldsymbol{f}}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{J_g}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi}\boldsymbol{J_g} \tag{5.64}$$

$$\approx \hat{\boldsymbol{e}}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{J_g} + \hat{\boldsymbol{e}}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{J_g}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi}\boldsymbol{J_g} \tag{5.65}$$

(drop $O(\varepsilon)$ terms from $\tilde{\boldsymbol{f}}$ in first summand to retain only $O(1)$ and $O(\beta\varepsilon)$) terms)

(drop $O(\varepsilon^2)$ terms from $\tilde{\boldsymbol{f}}\widetilde{\boldsymbol{J}}_{\boldsymbol{f}}$ in second summand)

$$= \hat{\boldsymbol{e}}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top (\boldsymbol{I} + \boldsymbol{J_g}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi})\boldsymbol{J_g} \tag{5.66}$$

and

$$\boldsymbol{J}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{a}\boldsymbol{g}^\top = \boldsymbol{J}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{J}_{\boldsymbol{g}}^\top (\boldsymbol{A}^{-1})^\top (\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top \tag{5.67}$$

$$\approx \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{J}_{\boldsymbol{g}}^\top (\boldsymbol{I} + \boldsymbol{J_g}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi})^\top (\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top \text{ (use first-order approx of } \boldsymbol{J_f}, \boldsymbol{A}) \tag{5.68}$$

$$\approx \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{J}_{\boldsymbol{g}}^\top (\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top \text{(drop } O(\varepsilon^2) \text{ terms from } \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}^{\boldsymbol{2}} \text{ in second summand)} \tag{5.69}$$

Or, more explicitly:

$$\frac{d\mathcal{L}}{d\boldsymbol{\xi}} = \boldsymbol{f}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{A}^{-1}\boldsymbol{J}_{\boldsymbol{g}} + \boldsymbol{J}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{J}_{\boldsymbol{g}}^\top (\boldsymbol{A}^{-1})^\top (\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top \tag{5.70}$$

$$\approx \hat{\boldsymbol{e}}(\nabla_{\boldsymbol{g}}\mathcal{L})^\top (\boldsymbol{I} + \boldsymbol{J}_{\boldsymbol{g}}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi})\boldsymbol{J}_{\boldsymbol{g}} + \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}^\top \boldsymbol{\xi}\boldsymbol{J}_{\boldsymbol{g}}^\top (\nabla_{\boldsymbol{g}}\mathcal{L})\boldsymbol{g}^\top \tag{5.71}$$

Analogously,

$$\frac{d\mathcal{L}}{d\beta} = (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \left(\boldsymbol{I} - \boldsymbol{J}_{\boldsymbol{g}}\boldsymbol{\xi}^\top \boldsymbol{J}_{\boldsymbol{f}}\boldsymbol{\xi}\right)^{-1} \boldsymbol{J}_{\boldsymbol{g}}\boldsymbol{\xi}^\top \left(\frac{1}{\beta}\boldsymbol{J}_{\boldsymbol{f}}\boldsymbol{h}\right) \tag{5.72}$$

$$\approx (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \left(\boldsymbol{I} + \boldsymbol{J}_{\boldsymbol{g}}\boldsymbol{\xi}^\top \widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\xi}\right) \boldsymbol{J}_{\boldsymbol{g}}\boldsymbol{\xi}^\top \left(\frac{1}{\beta}\widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{h}\right) \tag{5.73}$$

$$\approx (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{J}_{\boldsymbol{g}}\boldsymbol{\xi}^\top \left(\frac{1}{\beta}\widetilde{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{h}\right) \tag{5.74}$$

$$= (\nabla_{\boldsymbol{g}}\mathcal{L})^\top \boldsymbol{J}_{\boldsymbol{g}}\boldsymbol{\xi}^\top \left[\mathrm{diag}(\boldsymbol{\varepsilon}) - \boldsymbol{\varepsilon}\hat{\boldsymbol{e}}^\top - \hat{\boldsymbol{e}}\boldsymbol{\varepsilon}^\top + \left(\sum_\nu \varepsilon_\nu\right)\hat{\boldsymbol{e}}\hat{\boldsymbol{e}}^\top\right]\boldsymbol{h} \tag{5.75}$$

Note that in this case the first-order correction is $O(\varepsilon)$ rather than $O(\beta\varepsilon)$ as for $\frac{d\mathcal{L}}{d\boldsymbol{\xi}}$

5.3.4 Time and memory complexity

We assume naive matrix multiplication algorithm, which uses $O(MNP)$ time to multiply a $M \times N$ with an $N \times P$ matrix. Note that batching increases each computation by a factor of B (batch size), but the ratios remain unchanged.

Backpropagation through time

One update of the dynamics takes $O(MN)$ time to compute the hidden unit activities, and $O(NM)$ to compute the feature neuron activities, so the total time of the feedforward pass is $O(TMN)$. Similarly, the backwards pass is also $O(TMN)$.

To compute the backwards pass, we need to store the feature and hidden activity for every datapoint at every point in time. To store the activity for one point in time, for one datapoint $O(M + N)$ memory is required. To store this for all timepoints requires $O(T(M + N))$ memory.

Recurrent backpropagation

As before, the forward pass is $O(TMN)$ time. However, instead of backpropagating errors to compute the gradient, we evaluate the closed-form expression in Eq. 5.11. Assuming a naive cubic-time matrix inversion algorithm, inverting the $M \times M$ matrix $\boldsymbol{A}$ will take $O(M^3)$ time, so the total time to compute the gradient is $O(TMN + M^3)$. Thus, comparing to BPTT, the RBP algorithm has better time complexity if $TN > M^2$.

Since we do not need to store the neuron activity for intermediate timepoints, the memory required for the forward pass is $O(M + N)$. Inverting $\boldsymbol{A}$, however, naively requires $O(M^3)$ memory, resulting in a total memory complexity of $O(M^3 + N)$.

If we consider RBP-1 or RBP-1H, since we do not need to invert $\boldsymbol{A}$, evaluating the gradient is simply a small number of matrix multiplications taking $O(MN)$ time. Asymptotically, this is the same time complexity as BPTT, but for large T is be faster by almost a factor of 2 since we do not perform a full backwards pass. Similarly, the memory complexity is only $O(M + N)$.

Chapter 6: Conclusion

To summarize, in this thesis we have presented several neural network models which perform a range of memory tasks, demonstrating a variety of synaptic plasticity rules as well as memory addressing mechanisms. Chapter 1 presents an overview of memory addressing and draws an distinction between studies of plasticity and those of memory addressing. We emphasize that the rules which govern the selection of synapses to update, and those that define the updates themselves are separable, and may function together in various combinations.

In Chapter 2, we propose a neural network model of continual familiarity detection that uses strong feedforward weights for memory addressing and anti-Hebbian plasticity for storage. In this model, static feedforward synapses activate a set of neurons whose plastic synapses get *depressed* when pre- and post-synaptic neuron are co-active. As a result, when a familiar stimulus is presented, the neurons do not fire, resulting in the widely observed biological phenomenon of repetition suppression. Interestingly, these mechanisms were discovered by meta-learning – an optimization technique, rather than *a priori* ideas by the modeler.

In Chapter 3, we consider the problem of memory recall. Here, addressing for storage is achieved through local third factors. These can randomly or systematically activate neurons whose afferent synapses are updated to store a "key", and whose efferent synapses store its corresponding "value". Third factors can be generated intrinsically by neurons, or an external gating circuit. A plausible biological candidate for such third factors are dendritic plateau potentials – long-lasting depolarizations in the apical dendrites of pyramidal neurons which have been shown to gate the formation of place cells. The update rule is a "neo-Hebbian" three-factor rule which incorporates not only pre- and post-synaptic terms but also the third factor. For readout, on the other hand, memory addressing is achieved through content-based addressing. Given a query, the network selects the closest matching stored key through feedforward activations and returns its corresponding stored

value.

In Chapter 4, we consider a more challenging question – memory consolidation. Specifically, how it can occur in parallel with the process of memorization in a single system. Using the modern Hopfield network as the network architecture, we train it with gradient descent for autoassociative recall on a dataset consisting of items from a commonly-occurring class and a rare one. Common items should be consolidated into a general representation that ignores the fine details of specific items and rare ones should be memorized verbatim. To achieve this, we tune the learning rate or the sampling probability based on whether an item is represented in a distributed or a localized manner. Distributed items should be stored incrementally across many neurons and localized items should be rapidly memorized. In this way, both storage and recall are done in a content-addressed manner, but the synaptic update is tuned according to the statistics of the address.

Finally, Chapter 5 considers the gradient-based synaptic update process itself. In artificial recurrent neural networks, the gradient is computed with the biologically unrealistic algorithm of backpropagation through time. Here, we derive a version of recurrent backpropagation – originally proposed to compute the gradient for training fully connected recurrent attractor networks – for the modern Hopfield network, as well as a first-order approximation and a heuristic refinement, and suggest a biologically plausible interpretation.

Overall, we hope that this work highlights the distinction between two complementary mechanisms of memory storage – synaptic plasticity and memory addressing – and serves to inspire further theoretical and experimental work to further understand their mechanisms both separately and in union.

References

[1] A. Friedman *et al.*, "Analysis of complex neural circuits with nonlinear multidimensional hidden state models," *Proceedings of the National Academy of Sciences*, vol. 113, no. 23, pp. 6538–6543, 2016.

[2] D. Tyulmankov, G. R. Yang, and L. F. Abbott, "Meta-learning synaptic plasticity and memory addressing for continual familiarity detection," *Neuron*, vol. 110, no. 3, 544–557.e8, Feb. 2022.

[3] H. K. Titley, N. Brunel, and C. Hansel, "Toward a neurocentric view of learning," *Neuron*, vol. 95, no. 1, 19–32, Jul. 2017.

[4] N. Zucchet, S. Schug, J. von Oswald, D. Zhao, and J. Sacramento, "A contrastive rule for meta-learning," *Advances in Neural Information Processing Systems*, vol. 35, 25921–25936, Dec. 2022.

[5] R. Chaudhuri and I. Fiete, "Computational principles of memory," *Nature Neuroscience*, vol. 19, no. 33, 394–403, Mar. 2016.

[6] L. Abbott and W. G. Regehr, "Synaptic computation," *Nature*, vol. 431, no. 7010, pp. 796–803, 2004.

[7] D. S. Roy, A. Arons, T. I. Mitchell, M. Pignatelli, T. J. Ryan, and S. Tonegawa, "Memory retrieval by activating engram cells in mouse models of early alzheimer's disease," *Nature*, vol. 531, no. 75957595, 508–512, Mar. 2016.

[8] A. Treves and E. T. Rolls, "What determines the capacity of autoassociative memories in the brain?" *Network: Computation in Neural Systems*, vol. 2, no. 4, p. 371, 1991.

[9] S. Fusi, "Memory capacity of neural network models," no. arXiv:2108.07839, Dec. 2021, arXiv:2108.07839 [q-bio].

[10] D. Tarnoff, *Computer Organization and Design Fundamentals*. tarnoff, Jul. 2007, ISBN: 978-1-4116-3690-3.

[11] J. P. Eckert, "A survey of digital computer memory systems," *Proceedings of the IRE*, vol. 41, no. 10, 1393–1406, Oct. 1953.

[12] *Dynamic random access memory (dram). part 2: Read and write cycles*, `https://www.youtube.com/watch?v=x3jGqOrXXc8&list=PLTd6ceoshprfg23JMtwGysCm4tlc0I1ou&index=12`, Accessed 2023-10-10, YouTube.

[13] A. Citri and R. C. Malenka, "Synaptic plasticity: Multiple forms, functions, and mechanisms," *Neuropsychopharmacology*, vol. 33, no. 1, pp. 18–41, 2008.

[14] G. G. Turrigiano and S. B. Nelson, "Homeostatic plasticity in the developing nervous system," *Nature reviews neuroscience*, vol. 5, no. 2, p. 97, 2004.

[15] A. Statman, M. Kaufman, A. Minerbi, N. E. Ziv, and N. Brenner, "Synaptic size dynamics as an effectively stochastic process," *PLoS computational biology*, vol. 10, no. 10, e1003846, 2014.

[16] L. F. Abbott and S. B. Nelson, "Synaptic plasticity: Taming the beast," *Nature Neuroscience*, vol. 3, no. 1111, 1178–1183, Nov. 2000.

[17] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. New York: Wiley, 1949, vol. 65, ISBN: 978-1-4106-1240-3.

[18] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, no. 3, 267–273, Nov. 1982.

[19] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex," *Journal of Neuroscience*, vol. 2, no. 1, 32–48, Jan. 1982.

[20] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, "Eligibility traces and plasticity on behavioral time scales: Experimental support of neohebbian three-factor learning rules," *Frontiers in Neural Circuits*, vol. 12, 2018.

[21] D. Tyulmankov, C. Fang, A. Vadaparty, and G. R. Yang, "Biological learning in key-value memory networks," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, 22247–22258.

[22] S. Song and L. F. Abbott, "Cortical development and remapping through spike timing-dependent plasticity," *Neuron*, vol. 32, no. 2, 339–350, Oct. 2001.

[23] J.-P. Pfister and W. Gerstner, "Triplets of spikes in a model of spike timing-dependent plasticity," *Journal of Neuroscience*, vol. 26, no. 38, 9673–9682, Sep. 2006.

[24] M. Graupner and N. Brunel, "Calcium-based plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location," *Proceedings of the National Academy of Sciences*, vol. 109, no. 10, 3991–3996, Mar. 2012.

[25] K. Jacquerie, "Modeling brain-state dependent memory consolidation," Jul. 2023.

[26] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, 533–536, Oct. 1986.

[28] M. Akrout, C. Wilson, P. Humphreys, T. Lillicrap, and D. B. Tweed, "Deep learning without weight transport," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[29] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Communications*, vol. 7, p. 13 276, Nov. 2016.

[30] J. Guerguiev, T. P. Lillicrap, and B. A. Richards, "Towards deep learning with segregated dendrites," *eLife*, vol. 6, Dec. 2017.

[31] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 66, 335–346, Jun. 2020.

[32] T. P. Lillicrap and A. Santoro, "Backpropagation through time and the brain," *Current Opinion in Neurobiology*, Machine Learning, Big Data, and Neuroscience, vol. 55, 82–89, Apr. 2019.

[33] G. Bellec *et al.*, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Communications*, vol. 11, no. 11, p. 3625, Jul. 2020.

[34] Y. H. Liu, S. Smith, S. Mihalas, E. Shea-Brown, and U. Sümbül, "Biologically-plausible backpropagation through arbitrary timespans via local neuromodulators," *Advances in Neural Information Processing Systems*, vol. 35, 17528–17542, Dec. 2022.

[35] O. Marschall, K. Cho, and C. Savin, "Using local plasticity rules to train recurrent neural networks," no. arXiv:1905.12100, May 2019, arXiv:1905.12100 [cs, q-bio].

[36] J. M. Murray, "Local online learning in recurrent networks with random feedback," *eLife*, vol. 8, P. Latham, M. J. Frank, and B. DePasquale, Eds., e43299, May 2019.

[37] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," *Proceedings of IEEE International Conference on Neural Networks*, p. 10, 1987.

[38] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical Review Letters*, vol. 59, no. 19, 2229–2232, Nov. 1987.

[39] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation*. Redwood City, Calif.: Addison-Wesley Pub. Co., 1991, ISBN: 978-0-429-49966-1.

[40] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[41] B. Scellier and Y. Bengio, "Equivalence of equilibrium propagation and recurrent backpropagation," *Neural Computation*, vol. 31, no. 2, 312–329, Feb. 2019.

[42] D. Krotov and J. Hopfield, "Large associative memory problem in neurobiology and machine learning," *arXiv:2008.06996 [cond-mat, q-bio, stat]*, Mar. 2021, arXiv: 2008.06996.

[43] M. K. Benna and S. Fusi, "Computational principles of synaptic memory consolidation," *Nature Neuroscience*, vol. 19, no. 12, 1697–1706, Dec. 2016.

[44] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, 3521–3526, Mar. 2017.

[45] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," *arXiv:1703.04200 [cs, q-bio, stat]*, Mar. 2017, arXiv: 1703.04200.

[46] S. Lim *et al.*, "Inferring learning rules from distributions of firing rates in cortical neurons," *Nature Neuroscience*, vol. 18, no. 12, 1804–1810, Dec. 2015.

[47] Y. Mehta, D. Tyulmankov, A. E. Rajagopalan, G. C. Turner, J. E. Fitzgerald, and J. Funke, "Model-based inference of synaptic plasticity rules," *bioRxiv*, Dec. 2023.

[48] S. Fusi and L. F. Abbott, "Limits on the memory storage capacity of bounded synapses," *Nature Neuroscience*, vol. 10, no. 4, 485–493, Apr. 2007.

[49] S. Fusi, P. J. Drew, and L. Abbott, "Cascade models of synaptically stored memories," *Neuron*, vol. 45, no. 4, 599–611, Feb. 2005.

[50] J. Lindsey and A. Litwin-Kumar, *Theory of systems memory consolidation via recall-gated plasticity*. Dec. 2022.

[51] S. Lahiri and S. Ganguli, "A memory frontier for complex synapses," in *Advances in Neural Information Processing Systems*, vol. 26, Curran Associates, Inc., 2013.

[52] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, 115–133, Dec. 1943.

[53] E Gardner and B Derrida, "Optimal storage properties of neural network models," *Journal of Physics A: Mathematical and General*, vol. 21, no. 1, 271–284, Jan. 1988.

[54] A. B. Barrett and M. C. W. v. Rossum, "Optimal learning rules for discrete synapses," *PLOS Computational Biology*, vol. 4, no. 11, e1000230, Nov. 2008.

[55] M. C. W. Van Rossum, M. Shippi, and A. B. Barrett, "Soft-bound synaptic plasticity increases storage capacity," *PLoS Computational Biology*, vol. 8, no. 12, P. E. Latham, Ed., e1002836, Dec. 2012.

[56] R. Bogacz and M. W. Brown, "The restricted influence of sparseness of coding on the capacity of familiarity discrimination networks," p. 29, 2002.

[57] L. A. Jaeckel, *An alternative design for a sparse distributed memory*. Jul. 1989, NTRS Author Affiliations: Research Inst. for Advanced Computer ScienceNTRS Document ID: 19920001073NTRS Research Center: Legacy CDMS (CDMS).

[58] P. Kanerva, *Sparse Distributed Memory*. MIT Press, 1988, ISBN: 978-0-262-51469-9.

[59] P. Kanerva, *Sparse distributed memory and related models*. Apr. 1992, NTRS Author Affiliations: Research Inst. for Advanced Computer ScienceNTRS Document ID: 19920021480NTRS Research Center: Legacy CDMS (CDMS).

[60] P. Kanerva, *Sparse distributed memory and related models*. NASA Ames Research Center, Research Institute for Advanced Computer Science, 1992, vol. 92.

[61] L. A. Jaeckel, *A class of designs for a sparse distributed memory*. Jul. 1989, NTRS Author Affiliations: Research Inst. for Advanced Computer ScienceNTRS Document ID: 19920002426NTRS Research Center: Legacy CDMS (CDMS).

[62] A. G. Hanlon, "Content-addressable and associative memory systems a survey," *IEEE Transactions on Electronic Computers*, vol. EC-15, no. 4, 509–521, Aug. 1966.

[63] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, "Emerging trends in design and applications of memory-based computing and content-addressable memories," *Proceedings of the IEEE*, vol. 103, no. 8, 1311–1330, Aug. 2015.

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

[65] 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, 3088–3092, May 1984.

[66] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 1, 49–60, Feb. 1988.

[67] M. Demircigil, J. Heusel, M. Löwe, S. Upgang, and F. Vermet, "On a model of associative memory with huge storage capacity," *Journal of Statistical Physics*, vol. 168, no. 2, 288–299, Jul. 2017.

[68] D. Krotov and J. J. Hopfield, "Dense associative memory for pattern recognition," *arXiv:1606.01164 [cond-mat, q-bio, stat]*, Sep. 2016, arXiv: 1606.01164.

[69] H. Ramsauer *et al.*, "Hopfield networks is all you need," *arXiv:2008.02217 [cs, stat]*, Jul. 2020, arXiv: 2008.02217.

[70] D. Krotov and J. Hopfield, "Large associative memory problem in neurobiology and machine learning," *arXiv:2008.06996 [cond-mat, q-bio, stat]*, Mar. 2, 2021. arXiv: 2008.06996.

[71] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[72] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[73] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

[74] R. Bogacz, M. W. Brown, and C. Giraud-Carrier, "Model of familiarity discrimination in the perirhinal cortex," *Journal of Computational Neuroscience*, vol. 10, no. 1, 5–23, Jan. 2001.

[75] J. Wang, "Analysis and design of a k-winners-take-all model with a single state variable and the heaviside step activation function," *IEEE Transactions on Neural Networks*, vol. 21, no. 9, 1496–1506, Sep. 2010.

[76] E. Majani, R. Erlanson, and Y. Abu-Mostafa, "On the k-winners-take-all network," in *Advances in Neural Information Processing Systems*, vol. 1, Morgan-Kaufmann, 1988.

[77] K. A. Norman and R. C. O'Reilly, "Modeling hippocampal and neocortical contributions to recognition memory: A complementary-learning-systems approach," *Psychological Review*, vol. 110, no. 4, 611–646, 2003.

[78] R. C. O'Reilly and J. L. McClelland, "Hippocampal conjunctive encoding, storage, and recall: Avoiding a trade-off," *Hippocampus*, vol. 4, no. 6, 661–682, 1994.

[79] M. A. Snow and J. Orchard, "Biological softmax: Demonstrated in modern hopfield networks," *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 44, no. 44, 2022.

[80] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, Sep. 2014.

[81] A. Vaswani *et al.*, "Attention is all you need," *arXiv:1706.03762 [cs]*, Dec. 2017, arXiv: 1706.03762.

[82] Kuśmierz, T. Isomura, and T. Toyoizumi, "Learning with three factors: Modulating hebbian plasticity with errors," *Current Opinion in Neurobiology*, Computational Neuroscience, vol. 46, 170–177, Oct. 2017.

[83] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in Neural Circuits*, vol. 9, 2016.

[84] D. Foster, R. Morris, and P. Dayan, "A model of hippocampally dependent navigation, using the temporal difference learning rule," *Hippocampus*, vol. 10, no. 1, 1–16, 2000.

[85] T. Miconi, A. Rawal, J. Clune, and K. O. Stanley, "Backpropamine: Training self-modifying neural networks with differentiable neuromodulated plasticity," p. 15, 2019.

[86] K. C. Bittner, A. D. Milstein, C. Grienberger, S. Romani, and J. C. Magee, "Behavioral time scale synaptic plasticity underlies ca1 place fields," *Science*, vol. 357, no. 6355, 1033–1036, Sep. 2017.

[87] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proceedings of The 33rd International Conference on Machine Learning*, PMLR, Jun. 2016, 1842–1850.

[88] M Herrmann, J. A. Hertz, and A Prügel-Bennett, "Analysis of synfire chains," *Network: Computation in Neural Systems*, vol. 6, no. 3, 403–414, Jan. 1995.

[89] D. Tyulmankov, G. R. Yang, and L. Abbott, *Meta-learning hebbian plasticity for continual familiarity detection*, Poster presented at Computational and Systems Neuroscience (COSYNE) 2020, doi:10.7916/ytpp-sw91, 2020.

[90] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385.

[91] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, 1735–1780, Nov. 1997.

[92] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980 [cs]*, Jan. 2017, arXiv: 1412.6980.

[93] B. A. Richards and T. P. Lillicrap, "Dendritic solutions to the credit assignment problem," *Current Opinion in Neurobiology*, vol. 54, 28–36, Feb. 2019.

[94] P. R. Roelfsema and A. Holtmaat, "Control of synaptic plasticity in deep cortical networks," *Nature Reviews Neuroscience*, vol. 19, no. 33, 166–180, Mar. 2018.

[95] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, Feb. 22, 2021.

[96] D. Arpit *et al.*, "A closer look at memorization in deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, PMLR, Jul. 2017, 233–242.

[97] C. Stephenson, S. Padhy, A. Ganesh, Y. Hui, H. Tang, and S. Chung, "On the geometry of generalization and memorization in deep neural networks," no. arXiv:2105.14602, May 2021, arXiv:2105.14602 [cond-mat, stat].

[98] V. Feldman and C. Zhang, "What neural networks memorize and why: Discovering the long tail via influence estimation," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, 2881–2891.

[99] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, New York, NY, USA: Association for Computing Machinery, Oct. 2015, 1322–1333, ISBN: 978-1-4503-3832-5.

[100] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv:1312.6114 [cs, stat]*, May 2014, arXiv: 1312.6114.

[101] I. J. Goodfellow *et al.*, "Generative adversarial networks," no. arXiv:1406.2661, Jun. 2014, arXiv:1406.2661 [cs, stat].

[102] L. Yang *et al.*, "Diffusion models: A comprehensive survey of methods and applications," *ACM Computing Surveys*, vol. 56, no. 4, pp. 1–39, 2023.

[103] W. X. Zhao *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[104] C.-Y. Bai, H.-T. Lin, C. Raffel, and W. C.-w. Kan, "On training sample memorization: Lessons from benchmarking generative modeling with a large-scale competition," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Virtual Event Singapore: ACM, Aug. 2021, 2534–2542, ISBN: 978-1-4503-8332-5.

[105] S. Biderman *et al.*, "Emergent and predictable memorization in large language models," no. arXiv:2304.11158, Apr. 2023, arXiv:2304.11158 [cs].

[106] G. van den Burg and C. Williams, "On memorization in probabilistic deep generative models," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, 27916–27928.

[107] N. Carlini, D. Ippolito, M. Jagielski, K. Lee, F. Tramer, and C. Zhang, "Quantifying memorization across neural language models," no. arXiv:2202.07646, Mar. 2023, arXiv:2202.07646 [cs].

[108] K. Tirumala, A. Markosyan, L. Zettlemoyer, and A. Aghajanyan, "Memorization without overfitting: Analyzing the training dynamics of large language models," *Advances in Neural Information Processing Systems*, vol. 35, 38274–38290, Dec. 2022.

[109] I. Gulrajani, C. Raffel, and L. Metz, "Towards gan benchmarks which require generalization," no. arXiv:2001.03653, Jan. 2020, arXiv:2001.03653 [cs, stat].

[110] C. Meehan, K. Chaudhuri, and S. Dasgupta, "A non-parametric test to detect data-copying in generative models," no. arXiv:2004.05675, Apr. 2020, arXiv:2004.05675 [cs, stat].

[111] R. Webster, J. Rabin, L. Simon, and F. Jurie, "Detecting overfitting of deep generative networks via latent recovery," 2019, 11273–11282.

[112] A. Bai, C.-J. Hsieh, W. Kan, and H.-T. Lin, "Reducing training sample memorization in gans by training with memorization rejection," no. arXiv:2210.12231, Oct. 2022, arXiv:2210.12231 [cs].

[113] M. Widrich *et al.*, "Modern hopfield networks and attention for immune repertoire classification," *arXiv:2007.13505 [cs, q-bio, stat]*, Jul. 2020, arXiv: 2007.13505.

[114] D. Tyulmankov, K. Stachenfeld, D. Krotov, and L. Abbott, "Memorization and consolidation in associative memory networks," in *Associative Memory & Hopfield Networks in 2023*, 2023.

[115] S. Sukhbaatar, a. szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[116] J. Weston, S. Chopra, and A. Bordes, "Memory networks," no. arXiv:1410.3916, Nov. 2015, arXiv:1410.3916 [cs, stat].

[117] M. Fortunato *et al.*, "Generalization of reinforcement learners with working and episodic memory," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[118] C.-C. Hung *et al.*, "Optimizing agent behavior over long time scales by transporting value," *Nature Communications*, vol. 10, no. 11, p. 5223, Nov. 2019.

[119] G. Wayne *et al.*, "Unsupervised predictive memory in a goal-directed agent," no. arXiv:1803.10760, Mar. 2018, arXiv:1803.10760 [cs, stat].

[120] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[121] M. K. Benna and S. Fusi, "Place cells may simply be memory cells: Memory compression leads to spatial tuning and history dependence," *Proceedings of the National Academy of Sciences*, vol. 118, no. 51, Dec. 2021.

[122] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[123] L. Ambrogioni, "In search of dispersed memories: Generative diffusion models are associative memory networks," no. arXiv:2309.17290, Sep. 2023, arXiv:2309.17290 [cs, stat].

[124] B. Hoover, H. Strobelt, D. Krotov, J. Hoffman, Z. Kira, and D. H. Chau, "Memory in plain sight: A survey of the uncanny resemblances between diffusion models and associative memories," no. arXiv:2309.16750, Sep. 2023, arXiv:2309.16750 [cs, math].

[125] A. Radhakrishnan, M. Belkin, and C. Uhler, "Overparameterized neural networks implement associative memory," *Proceedings of the National Academy of Sciences*, vol. 117, no. 44, 27162–27170, Nov. 2020.

[126] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[127] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv:1410.5401 [cs]*, Oct. 2014, arXiv: 1410.5401.

[128] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, 471–476, Oct. 2016.

[129] W. Zhang, Y. Yu, and B. Zhou, "Structured memory for neural turing machines," *arXiv:1510.03931 [cs]*, Oct. 2015, arXiv: 1510.03931.

[130] J. Rae *et al.*, "Scaling memory-augmented neural networks with sparse reads and writes," in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016.

[131] J. Ba, G. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu, "Using fast weights to attend to the recent past," *arXiv:1610.06258 [cs, stat]*, Oct. 2016, arXiv: 1610.06258.

[132] T. Miconi, J. Clune, and K. O. Stanley, "Differentiable plasticity: Training plastic neural networks with backpropagation," *arXiv:1804.02464 [cs, stat]*, Apr. 2018, arXiv: 1804.02464.

[133] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine Learning*, vol. 7, no. 2, 195–225, Sep. 1991.

[134] J. Collins, J. Sohl-Dickstein, and D. Sussillo, "Capacity and trainability in recurrent neural networks," *arXiv:1611.09913 [cs, stat]*, Nov. 2016, arXiv: 1611.09913.

[135] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," no. arXiv:1406.1078, Sep. 2014, arXiv:1406.1078 [cs, stat].

[136] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," no. arXiv:1506.02078, Nov. 2015, arXiv:1506.02078 [cs].

[137] Z. Ashwood, N. A. Roy, J. H. Bak, and J. W. Pillow, "Inferring learning rules from animal decision-making," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, 3442–3453.

[138] K. S. Lashley *et al.*, *The problem of serial order in behavior*. Bobbs-Merrill Oxford, 1951, vol. 21.

[139] R. W. Semon, *The mneme*. G. Allen & Unwin Limited, 1921.

[140] P. W. Frankland, S. A. Josselyn, and S. Köhler, "The neurobiological foundation of memory retrieval," *Nature Neuroscience*, vol. 22, no. 1010, 1576–1585, Oct. 2019.

[141] S. A. Josselyn and S. Tonegawa, "Memory engrams: Recalling the past and imagining the future," *Science*, vol. 367, no. 6473, eaaw4325, Jan. 2020.

[142] X. Liu *et al.*, "Optogenetic stimulation of a hippocampal engram activates fear memory recall," *Nature*, vol. 484, no. 7394, pp. 381–385, 2012.

[143] C. A. Denny *et al.*, "Hippocampal memory traces are differentially modulated by experience, time, and adult neurogenesis," *Neuron*, vol. 83, no. 1, pp. 189–201, 2014.

[144] S. A. Josselyn, S. Köhler, and P. W. Frankland, "Finding the engram," *Nature Reviews Neuroscience*, vol. 16, no. 99, 521–534, Sep. 2015.

[145] M. Kim *et al.*, "Functional and topological conditions for explosive synchronization develop in human brain networks with the onset of anesthetic-induced unconsciousness," *Frontiers in Computational Neuroscience*, vol. 10, Jan. 2016.

[146] T. J. Teyler and P. DiScenna, "The hippocampal memory indexing theory," *Behavioral Neuroscience*, vol. 100, no. 2, 147–154, 1986.

[147] T. J. Teyler and J. W. Rudy, "The hippocampal indexing theory and episodic memory: Updating the index," *Hippocampus*, vol. 17, no. 12, 1158–1169, 2007.

[148] D. Krotov, "Hierarchical associative memory," *arXiv:2107.06446 [cs]*, Jul. 2021, arXiv: 2107.06446.

[149] T. D. Goode, K. Z. Tanaka, A. Sahay, and T. J. McHugh, "An integrated index: Engrams, place cells, and hippocampal memory," *Neuron*, vol. 107, no. 5, 805–820, Sep. 2020.

[150] D. S. Roy *et al.*, "Brain-wide mapping reveals that engrams for a single memory are distributed across multiple brain regions," *Nature communications*, vol. 13, no. 1, p. 1799, 2022.

[151] K. Z. Tanaka, H. He, A. Tomar, K. Niisato, A. J. Huang, and T. J. McHugh, "The hippocampal engram maps experience but not place," *Science*, vol. 361, no. 6400, pp. 392–397, 2018.

[152] S. N. Chettih, E. L. Mackevicius, S. Hale, and D. Aronov, "Barcoding of episodic memories in the hippocampus of a food-caching bird," *bioRxiv*, 2023.

[153] J. von Oswald *et al.*, "Learning where to learn: Gradient sparsity in meta and continual learning," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, 5250–5263.

[154] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 75537553, pp. 436–444, May 2015.

[155] G. Parisi, "A memory which forgets," *Journal of Physics A: Mathematical and General*, vol. 19, no. 10, pp. L617–L620, Jul. 1986.

[156] 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, Nov. 2013.

[157] U. Hasson, J. Chen, and C. J. Honey, "Hierarchical process memory: Memory as an integral component of information processing," *Trends in Cognitive Sciences*, vol. 19, no. 6, pp. 304–313, Jun. 2015.

[158] S. Beaulieu *et al.*, "Learning to continually learn," *arXiv:2002.09571 [cs, stat]*, Mar. 2020, arXiv: 2002.09571.

[159] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, Oct. 2016.

[160] L. Standing, "Learning 10000 pictures," *Quarterly Journal of Experimental Psychology*, vol. 25, no. 2, pp. 207–222, May 1973.

[161] R. Bogacz and M. W. Brown, "Comparison of computational models of familiarity discrimination in the perirhinal cortex," *Hippocampus*, vol. 13, no. 4, pp. 494–524, 2003.

[162] T. F. Brady, T. Konkle, G. A. Alvarez, and A. Oliva, "Visual long-term memory has a massive storage capacity for object details," *Proceedings of the National Academy of Sciences*, vol. 105, no. 38, pp. 14 325–14 329, Sep. 2008.

[163] L. Ji-An, F. Stefanini, M. K. Benna, and S. Fusi, "Face familiarity detection with complex synapses," *bioRxiv*, p. 854 059, Nov. 2019.

[164] K. Grill-Spector, R. Henson, and A. Martin, "Repetition and the brain: Neural models of stimulus-specific effects," *Trends in Cognitive Sciences*, vol. 10, no. 1, pp. 14–23, Jan. 2006.

[165] T. Meyer and N. C. Rust, "Single-exposure visual memory judgments are reflected in inferotemporal cortex," *eLife*, vol. 7, Mar. 2018.

[166] E. K. Miller, L. Li, and R. Desimone, "A neural mechanism for working and recognition memory in inferior temporal cortex," *Science*, vol. 254, no. 5036, pp. 1377–1379, 1991.

[167] J. Z. Xiang and M. W. Brown, "Differential neuronal encoding of novelty, familiarity and recency in regions of the anterior temporal lobe," *Neuropharmacology*, vol. 37, no. 4, pp. 657–676, Apr. 1998.

[168] Z. Androulidakis, A. Lulham, R. Bogacz, and M. W. Brown, "Computational models can replicate the capacity of human recognition memory," *Network: Computation in Neural Systems*, vol. 19, no. 3, pp. 161–182, Jan. 2008.

[169] K. A. Norman and R. C. O'Reilly, "Modeling hippocampal and neocortical contributions to recognition memory: A complementary-learning-systems approach," *Psychological Review*, vol. 110, no. 4, pp. 611–646, 2003.

[170] V. S. Sohal and M. E. Hasselmo, "A model for experience-dependent changes in the responses of inferotemporal neurons," *Network: Computation in Neural Systems*, vol. 11, no. 3, pp. 169–190, Jan. 2000.

[171] S. Thrun and L. Pratt, *Learning to Learn*. Springer Science & Business Media, Dec. 2012, Google-Books-ID: $X_jpBwAAQBAJ$, ISBN: 978-1-4615-5529-2.

[172] Y. Bengio, S. Bengio, and J. Cloutier, "Learning a synaptic learning rule," 1991.

[173] J. Jordan, M. Schmidt, W. Senn, and M. A. Petrovici, "Evolving to learn: Discovering interpretable plasticity rules for spiking networks," *arXiv:2005.14149 [q-bio]*, Jan. 2021, arXiv: 2005.14149.

[174] J. Lindsey and A. Litwin-Kumar, "Learning to learn with feedback and local plasticity," *arXiv:2006.09549 [cs, q-bio]*, Jun. 2020, arXiv: 2006.09549.

[175] K. Gu, S. Greydanus, L. Metz, N. Maheswaranathan, and J. Sohl-Dickstein, "Meta-learning biologically plausible semi-supervised update rules," *bioRxiv*, p. 2019.12.30.891184, Dec. 2019.

[176] B. Confavreux, E. J. Agnes, F. Zenke, T. Lillicrap, and T. P. Vogels, "A meta-learning approach to (re)discover plasticity rules that carve a desired function into a neural network," *bioRxiv*, p. 2020.10.24.353409, Oct. 2020.

[177] L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein, "Meta-learning update rules for unsupervised representation learning," *arXiv:1804.00222 [cs, stat]*, Feb. 2019, arXiv: 1804.00222.

[178] E. Najarro and S. Risi, "Meta-learning through hebbian plasticity in random networks," *arXiv:2007.02686 [cs]*, Mar. 2021, arXiv: 2007.02686.

[179] B. B. Murdock, "An analysis of the strength-latency relationship," *Memory & Cognition*, vol. 13, no. 6, pp. 511–521, Nov. 1985.

[180] S. R. Lehky and K. Tanaka, "Neural representation for object recognition in inferotemporal cortex," *Current Opinion in Neurobiology*, Neurobiology of cognitive behavior, vol. 37, pp. 23–35, Apr. 2016.

[181] A. Lueschow, E. K. Miller, and R. Desimone, "Inferior temporal mechanisms for invariant object recognition," *Cerebral Cortex*, vol. 4, no. 5, pp. 523–531, Sep. 1994.

[182] U. Rutishauser *et al.*, "Representation of retrieval confidence by single neurons in the human medial temporal lobe," *Nature Neuroscience*, vol. 18, no. 77, pp. 1041–1050, Jul. 2015.

[183] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255.

[184] M. Lundqvist, P. Herman, and E. K. Miller, "Working memory: Delay activity, yes! persistent activity? maybe not," *Journal of Neuroscience*, vol. 38, no. 32, pp. 7013–7019, Aug. 2018.

[185] N. Y. Masse, M. C. Rosen, and D. J. Freedman, "Reevaluating the role of persistent neural activity in short-term memory," *Trends in Cognitive Sciences*, vol. 24, no. 3, pp. 242–258, Mar. 2020.

[186] A. Schulz, C. Miehl, M. J. Berry, and J. Gjorgjieva, "The generation of cortical novelty responses through inhibitory plasticity," *bioRxiv*, p. 2020.11.30.403840, Dec. 2020.

[187] 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.

[188] D. J. Amit, *Modeling brain function: The world of attractor neural networks*. Cambridge university press, 1992.

[189] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," *Nature*, vol. 222, no. 5197, pp. 960–962, 1969.

[190] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," *arXiv preprint arXiv:1503.08895*, 2015.

[191] T. Munkhdalai, A. Sordoni, T. Wang, and A. Trischler, "Metalearned neural memory," *arXiv preprint arXiv:1907.09720*, 2019.

[192] H. Le, T. Tran, and S. Venkatesh, "Neural stored-program memory," *arXiv preprint arXiv:1906.08862*, 2019.

[193] S. Bartunov, J. W. Rae, S. Osindero, and T. P. Lillicrap, "Meta-learning deep energy-based memory models," *arXiv preprint arXiv:1910.02720*, 2019.

[194] N. Rodriguez, E. Izquierdo, and Y.-Y. Ahn, "Optimal modularity and memory capacity of neural reservoirs," *Network Neuroscience*, vol. 3, no. 2, pp. 551–566, Jan. 2019.

[195] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[196] A. Banino *et al.*, "Memo: A deep network for flexible combination of episodic memories," *arXiv preprint arXiv:2001.10913*, 2020.

[197] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *International Conference on Learning Representations*, 2015.

[198] A. Vaswani *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[199] J. Weston *et al.*, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[200] C.-C. Hung *et al.*, "Optimizing agent behavior over long time scales by transporting value," *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

[201] H. Ramsauer *et al.*, "Hopfield networks is all you need," *arXiv preprint arXiv:2008.02217*, 2020.

[202] D. Krotov and J. J. Hopfield, "Dense associative memory for pattern recognition," *arXiv preprint arXiv:1606.01164*, 2016.

[203] D. Krotov and J. Hopfield, "Large associative memory problem in neurobiology and machine learning," *arXiv preprint arXiv:2008.06996*, 2020.

[204] D. Krotov, "Hierarchical Associative Memory," *arXiv:2107.06446 [cs]*, Jul. 2021, arXiv: 2107.06446.

[205] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, "Eligibility traces and plasticity on behavioral time scales: Experimental support of neohebbian three-factor learning rules," *Frontiers in Neural Circuits*, vol. 12, p. 53, 2018.

[206] G. J. Stuart and N. Spruston, "Dendritic integration: 60 years of progress," *Nature neuroscience*, vol. 18, no. 12, pp. 1713–1721, 2015.

[207] F. Gambino *et al.*, "Sensory-evoked ltp driven by dendritic plateau potentials in vivo," *Nature*, vol. 515, no. 7525, pp. 116–119, 2014.

[208] D. Tyulmankov, G. R. Yang, and L. Abbott, "Meta-learning local synaptic plasticity for continual familiarity detection," *bioRxiv*, 2021.

[209] K. C. Bittner, A. D. Milstein, C. Grienberger, S. Romani, and J. C. Magee, "Behavioral time scale synaptic plasticity underlies ca1 place fields," *Science*, vol. 357, no. 6355, pp. 1033–1036, 2017.

[210] D. D. Rasmusson, "The role of acetylcholine in cortical synaptic plasticity," *Behav Brain Res.*, vol. 115, no. 2, pp. 205–218, 2000.

[211] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, Elsevier, 1989, pp. 109–165.

[212] A. Robins and S. McCallum, "Catastrophic forgetting and the pseudorehearsal solution in hopfield-type networks," *Connection Science*, vol. 10, no. 2, pp. 121–135, 1998.

[213] D. J. Amit, H. Gutfreund, and H. Sompolinsky, "Storing infinite numbers of patterns in a spin-glass model of neural networks," *Physical Review Letters*, vol. 55, no. 14, p. 1530, 1985.

[214] T. M. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Transactions on Electronic Computers*, vol. EC-14, no. 3, pp. 326–334, Jun. 1965, Conference Name: IEEE Transactions on Electronic Computers.

[215] E. Gardner, "The space of interactions in neural network models," *Journal of physics A: Mathematical and general*, vol. 21, no. 1, p. 257, 1988.

[216] M. Demircigil, J. Heusel, M. Löwe, S. Upgang, and F. Vermet, "On a Model of Associative Memory with Huge Storage Capacity," *Journal of Statistical Physics*, vol. 168, no. 2, pp. 288–299, Jul. 2017.

[217] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[218] G. Parisi, "A memory which forgets," *Journal of Physics A: Mathematical and General*, vol. 19, no. 10, p. L617, 1986.

[219] R. Brown and J. Kulik, "Flashbulb memories," *Cognition*, vol. 5, no. 1, pp. 73–99, 1977.

[220] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, man, and Cybernetics*, vol. 18, no. 1, pp. 49–60, 1988.

[221] D. J. Foster and M. A. Wilson, "Reverse replay of behavioural sequences in hippocampal place cells during the awake state," *Nature*, vol. 440, pp. 680–683, 2006.

[222] B. E. Pfeiffer and D. J. Foster, "Hippocampal place-cell sequences depict future paths to remembered goals," *Nature*, vol. 497, pp. 74–79, 2013.

[223] M. G. Mattar and N. D. Daw, "Prioritized memory access explains planning and hippocampal replay," *Nature Neuroscience*, vol. 21, pp. 1609–1617, 2018.

[224] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory," *Psychological Review*, vol. 102, pp. 419–457, 1995, Place: US Publisher: American Psychological Association.

[225] D. Kumaran, D. Hassabis, and J. L. McClelland, "What learning systems do intelligent agents need? complementary learning systems theory updated," *Trends in Cognitive Sciences*, vol. 20, no. 7, pp. 512–534, Jul. 1, 2016.

[226] J. M. Fulvio, C. S. Green, and P. R. Schrater, "Task-specific response strategy selection on the basis of recent training experience," *PLOS Computational Biology*, vol. 10, no. 1, e1003425, Jan. 2, 2014, Publisher: Public Library of Science.

[227] T. Davis, B. C. Love, and A. R. Preston, "Learning the exception to the rule: Model-based fMRI reveals specialized representations for surprising category members," *Cerebral Cortex*, vol. 22, no. 2, pp. 260–273, Feb. 2012.

[228] E. M. Heffernan, M. L. Schlichting, and M. L. Mack, "Learning exceptions to the rule in human and model via hippocampal encoding," *Scientific Reports (Nature Publisher Group)*, vol. 11, no. 1, 2021, Place: London, United States Publisher: Nature Publishing Group.

[229] L. M. Boyle, L. Posani, S. Irfan, S. A. Siegelbaum, and S. Fusi, *Tuned geometries of hippocampal representations meet the demands of social memory*, Pages: 2022.01.24.477361 Section: New Results, Feb. 15, 2023.

[230] 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, Apr. 1, 1982.

[231] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, Number: 7553 Publisher: Nature Publishing Group.

[232] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, p. 27, Mar. 19, 2019.

[233] L. Wittkuhn, S. Chien, S. Hall-McMaster, and N. W. Schuck, "Replay in minds and machines," *Neuroscience & Biobehavioral Reviews*, vol. 129, pp. 367–388, Oct. 1, 2021.

[234] D. Krotov and J. J. Hopfield, "Dense associative memory for pattern recognition," *arXiv:1606.01164 [cond-mat, q-bio, stat]*, Sep. 27, 2016. arXiv: 1606.01164.

[235] H. Ramsauer *et al.*, "Hopfield networks is all you need," *arXiv:2008.02217 [cs, stat]*, Jul. 16, 2020. arXiv: 2008.02217.

[236] T. P. Lillicrap and A. Santoro, "Backpropagation through time and the brain," *Current Opinion in Neurobiology*, Machine Learning, Big Data, and Neuroscience, vol. 55, pp. 82–89, Apr. 1, 2019.

[237] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 335–346, Jun. 2020, Number: 6 Publisher: Nature Publishing Group.

[238] G. Parisi, "A memory which forgets," *Journal of Physics A: Mathematical and General*, vol. 19, no. 10, pp. L617–L620, Jul. 1986.

[239] D. Tyulmankov, C. Fang, A. Vadaparty, and G. R. Yang, "Biological learning in key-value memory networks," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, pp. 22 247–22 258.

[240] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Computing Surveys*, vol. 54, no. 2, 38:1–38:38, Mar. 5, 2021.

[241] A. Boukerche, L. Zheng, and O. Alfandi, "Outlier detection: Methods, models, and classification," *ACM Computing Surveys*, vol. 53, no. 3, 55:1–55:37, Jun. 12, 2020.

[242] J. Zhang *et al.*, "Out-of-distribution detection based on in-distribution data patterns memorization with modern hopfield energy," presented at the The Eleventh International Conference on Learning Representations, Feb. 1, 2023.

[243] J. M. Murray and G. S. Escola, "Remembrance of things practiced with fast and slow learning in cortical and subcortical pathways," *Nature Communications*, vol. 11, no. 1, p. 6441, Dec. 23, 2020, Number: 1 Publisher: Nature Publishing Group.

[244] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Surveys*, vol. 53, no. 3, 63:1–63:34, Jun. 12, 2020.

[245] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, *Deep double descent: Where bigger models and more data hurt*, Dec. 4, 2019. arXiv: 1912.02292[cs,stat].

[246] C. Stephenson, S. Padhy, A. Ganesh, Y. Hui, H. Tang, and S. Chung, *On the geometry of generalization and memorization in deep neural networks*, May 30, 2021. arXiv: 2105.14602[cond-mat,stat].

[247] D. Patel and P. S. Sastry, "Memorization in deep neural networks: Does the loss function matter?" In *Advances in Knowledge Discovery and Data Mining*, K. Karlapalem *et al.*, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2021, pp. 131–142, ISBN: 978-3-030-75765-6.

[248] M. Bahrami, "RECOGNITION OF RULES AND EXCEPTIONS BY NEURAL NETWORKS," *International Journal of Neural Systems*, vol. 02, no. 4, pp. 341–344, Jan. 1991.

[249] M. Ruiz-Garcia, G. Zhang, S. S. Schoenholz, and A. J. Liu, "Tilting the playing field: Dynamical loss functions for machine learning," in *Proceedings of the 38th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Jul. 1, 2021, pp. 9157–9167.

[250] B. Sorscher, R. Geirhos, S. Shekhar, S. Ganguli, and A. S. Morcos, *Beyond neural scaling laws: Beating power law scaling via data pruning*, Nov. 15, 2022. arXiv: 2206.14486[cs,stat].

[251] V. Feldman and C. Zhang, "What neural networks memorize and why: Discovering the long tail via influence estimation," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 2881–2891.

[252] M. Carandini and D. J. Heeger, "Normalization as a canonical neural computation," *Nature Reviews Neuroscience*, vol. 13, no. 1, pp. 51–62, Jan. 2012, Number: 1 Publisher: Nature Publishing Group.

[253] Z.-H. Mao and S. G. Massaquoi, "Dynamics of winner-take-all competition in recurrent neural networks with lateral inhibition," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 55–69, Jan. 2007, Conference Name: IEEE Transactions on Neural Networks.

[254] M. A. Snow and J. Orchard, "Biological softmax: Demonstrated in modern hopfield networks," *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 44, no. 44, 2022.

[255] D. Krotov, "Hierarchical associative memory," *arXiv:2107.06446 [cs]*, Jul. 13, 2021. arXiv: 2107.06446.

[256] C. Lucibello and M. Mézard, "The exponential capacity of dense associative memories," *arXiv preprint arXiv:2304.14964*, 2023.

[257] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980 [cs]*, Jan. 29, 2017. arXiv: 1412.6980.

[258] O. Barak, M. Rigotti, and S. Fusi, "The sparseness of mixed selectivity neurons controls the generalization–discrimination trade-off," *Journal of Neuroscience*, vol. 33, no. 9, pp. 3844–3856, Feb. 27, 2013, Publisher: Society for Neuroscience Section: Articles.

[259] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv:1312.6114 [cs, stat]*, May 1, 2014. arXiv: 1312.6114.

[260] S. Biderman *et al.*, *Emergent and predictable memorization in large language models*, Apr. 21, 2023. arXiv: 2304.11158[cs].

[261] S. Bird *et al.*, "Fairlearn: A toolkit for assessing and improving fairness in AI," May 18, 2020.

[262] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," *Proceedings of IEEE International Conference on Neural Networks*, p. 10, 1987.

[263] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical Review Letters*, vol. 59, no. 19, pp. 2229–2232, Nov. 9, 1987, Publisher: American Physical Society.

[264] M. Widrich *et al.*, "Modern Hopfield Networks and Attention for Immune Repertoire Classification," *arXiv:2007.13505 [cs, q-bio, stat]*, Jul. 2020, arXiv: 2007.13505.

[265] R. Chaudhuri and I. Fiete, "Bipartite expander Hopfield networks as self-decoding high-capacity error correcting codes," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[266] N. Frémaux and W. Gerstner, "Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules," *Frontiers in Neural Circuits*, vol. 9, 2016.

[267] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, "Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules," *Frontiers in Neural Circuits*, vol. 12, 2018.

[268] J. C. R. Whittington and R. Bogacz, "Theories of Error Back-Propagation in the Brain," *Trends in Cognitive Sciences*, vol. 23, no. 3, pp. 235–250, Mar. 2019.

[269] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation*. Redwood City, Calif.: Addison-Wesley Pub. Co., 1991, OCLC: 645821440, ISBN: 978-0-429-49966-1.

[270] S. Bai, J. Z. Kolter, and V. Koltun, "Deep Equilibrium Models," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[271] R. Liao *et al.*, "Reviving and improving recurrent back-propagation," in *International Conference on Machine Learning*, PMLR, 2018, pp. 3082–3091.