On the Role of Attention in Modern Hopfield Network Models of the Hippocampus

Shaunak Bhandarkar
shaunakb@stanford.edu
Stanford University

1 Introduction

1.1 Some Motivation

The Complementary Learning Systems (CLS) Theory, first introduced by McClelland, McNaughton, and O'Reilly in the context of semantic cognition,[1] proposes that a slow-learning semantic knowledge system in the neocortex is complemented by a quick-learning system in the medial temporal lobes: the hippocampus (McClelland and Rogers, 2003).[2] The hippocampus rapidly learns from highly processed data, and the resulting hippocampal representations are gradually consolidated into the neocortex. To better understand the interplay between the hippocampus and cortical areas, it is crucial to understand how memories are stored in and retrieved from the hippocampus. The aim of this paper is to focus in on the hippocampal part of the CLS framework, exploring how hippocampal representations of semantic data arise. By using the Parallel Distributed Processing approach, I aim to answer the following key questions:

1. How (and how "quickly") can the hippocampus match incoming processed data to stored memories in order to most accurately evoke the stored memory that corresponds to the data?

2. How might the hippocampus support the rapid acquisition of new knowledge?

1.2 Introducing the Modern Hopfield Network

To model memory storage and retrieval, I use the Modern Hopfield Network, as described in Krotov and Hopfield (2021).[3] This network can be thought of as a two-layer network, with bidirectional connections between the two layers. One layer is a "feature layer", which contains the feature values for a given input pattern, and the other layer is the "memory layer", in which each neuron corresponds to a single stored memory. Intuitively, the Modern Hopfield Network repeatedly compares the similarity of its input pattern to the memories that it has stored and updates the input pattern correspondingly; at the end of the process, the activations in the feature layer should ideally have converged to the stored memory that most closely matches the original input pattern.

While Modern Hopfield Networks aren't exactly biologically plausible (one neuron is not enough to store a memory), their underlying process resembles that of the CA3 recurrent collaterals in the hippocampus. Sparsified representations from the dentate gyrus are passed through these collaterals, which effectively perform pattern completion to retrieve a "complete" hippocampal representation.

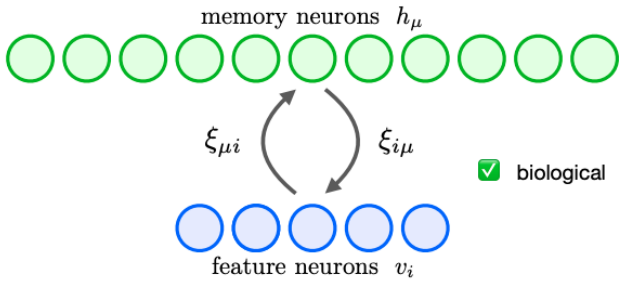


Figure 1. A diagram of a Modern Hopfield Network, as presented in Krotov and Hopfield (2021).[3] This network has $N_f = 5$ feature neurons (for taking in 5-dimensional input vectors) and $N_h = 11$ hidden memory neurons, that correspond to 11 stored memory vectors.

Mathematically, say we have N_h memories stored in our network: $v_1, v_2, \ldots, v_{N_h}$, and each $v_i \in \mathbb{R}^{N_f}$, where the number of neurons in the feature layer of the network is N_f. Letting $\xi_{\mu i}$ be the connection weight going from feature neuron i to memory neuron μ and $\xi_{i\mu}$ the connection weight going from memory neuron μ to feature neuron i, we set $\xi_{\mu i} = \xi_{i\mu} = v_\mu^i$, i.e. the ith component of the μth stored memory. In other words, the memories are literally hard-coded into the weights of the network.

The Modern Hopfield Network (generally) excels at pattern completion. If we pass an input feature vector $v^{(0)}$ into the network (that we assume is an "incomplete" partial representation of some memory) at time $t = 0$, the update rule of the network is given by

$$v_i^{(t+1)} = v_i^{(t)} + \frac{1}{\tau}\left(\sum_{\mu=1}^{N_h} \xi_{i\mu} \text{softmax}\left(\sum_{j=1}^{N_f} \xi_{\mu j} v_j^{(t)}\right) - v_i^{(t)}\right),$$

where τ denotes a temporal parameter (which we will set to 1 unless otherwise stated), and $v_i^{(t)}$ denotes the ith component of the vector v at time $t + 1$. Setting $\tau = 1$, we obtain

$$v_i^{(t+1)} = \sum_{\mu=1}^{N_h} \xi_{i\mu} \text{softmax}\left(\sum_{j=1}^{N_f} \xi_{\mu j} v_j^{(t)}\right).$$

This equation is none other than the *attention equation* for a Transformer! In other words, the Hopfield Net repeatedly uses self-attention to "complete" a partial memory representation into a "complete" memory representation.

1.3 The Approach of this Paper

In line with the two guiding questions presented at the start of the paper, the notion of attention is crucial to how accurately a Modern Hopfield Net is able to retrieve a full memory representation from a partial one. Interestingly, Krotov and Hopfield (2021)[3] do not consider the so-called *slope parameter* of the softmax function, which modulates the level of attention given to each of the stored memories in the network. Specifically, we can extend the softmax function to a new function that accepts a vector $x = (x_1, \ldots, x_N)$ *and* a slope parameter β as inputs:

$$\text{softmax}_\beta(x) = \frac{1}{\sum_{i=1}^{N} \exp(\beta x_i)}(\exp(\beta x_1), \ldots, \exp(\beta x_N)).$$

(Note that setting $\beta = 1$ yields the ordinary softmax function.) In this paper, I first investigate the performance of a Modern Hopfield Network that is equipped with an additional slope parameter β, as described above. By varying the value of β, I observe how the "level of attention" across all stored memories can affect retrieval accuracy.

A big limitation of the Modern Hopfield Network is that it stores a fixed number of memories, N_h. Therefore, another central approach used in this paper is to design a "Fluid Hopfield Network" that is capable of acquiring new memories and storing them in the memory layer (meaning that N_h gets larger). However, to most reasonably simulate biological acquisition of memories, any newly acquired memories should start off with only small amount of attention given to them; more rigorously, there should be a unique slope parameter for *each* memory that either increases (resp. decreases) based on repeated exposure to similar (resp. dissimilar) feature patterns. In other words, the process of giving attention to each of the stored memories should be automated. Mathematically, we can let σ_μ denote the slope parameter for the μth memory, and extend the softmax function to a generalized softmax function

$$\text{softmax}_{(\sigma_1,\ldots,\sigma_N)}(x) = \frac{1}{\sum_{j=1}^{N} \exp(\sigma_j x_j)}(\exp(\sigma_1 x_1), \ldots, \exp(\sigma_N x_N)),$$

where each σ_j is updated at each iteration of the Modern Hopfield Network process. For example, if the network newly acquires the feature pattern corresponding to 'lizard', the network should not immediately be expected to retrieve the feature pattern for 'lizard'; indeed, the newly assigned slope parameter to 'lizard' will be quite small. However, perhaps the next time we pass in the pattern corresponding to a 'lizard' (or even a partial pattern of the feature pattern corresponding to a 'lizard'), the network's slope parameter for 'lizard' will have increased, enabling the network to accurately retrieve this newly acquired memory of a 'lizard'. From a biological standpoint, we should think of σ_μ as a sort of "synaptic strength" that reflects how well-consolidated the μth memory of the network is; increasing σ_μ is analogous to long-term potentiation.

Another central tenet of this paper is to understand how the extended model mirrors existing neurological conditions pertaining to the hippocampus. To this end, I will examine how speci-

fying certain starting parameters for the "Fluid Hopfield Net" can cause the network to exhibit characteristic traits of different kinds of amnesia – specifically, semantic amnesia and anterograde amnesia.

2 Methods

In studying the storage and retrieval processes for semantic memory, the "memories" that will be stored in my Modern Hopfield Network will come from the McClelland and Rogers (2003)[2] semantic cognition dataset, which consists of combinations of several items and relations (e.g. "pine IS ", "salmon HAS ", "canary IS A ", etc.) and lists the multi-hot feature vectors for these item-relation combinations. For this paper, I use this dataset except without the relations; rather, each item (e.g. canary, salmon, oak, etc.) is specified by a feature vector that contains the features of that item across *all of the relations*. For instance, for the canary's feature vector, the 'yellow' slot will have a 1, the 'wings' slot will have a 1, the 'animal' slot will have a 1, the 'flies' slot will have a 1, and so on; features that do not describe the canary will have a 0 in their corresponding slots. There are 8 items in the dataset: pine, oak, rose, daisy, robin, canary, sunfish, and salmon. Each item has a feature 35-dimensional feature vector (which is slightly modified from the original semantic cognition dataset, which had 36-dimensional vectors; I removed the slightly redundant feature 'living'). The job of the Modern Hopfield Network is to take a partial feature vector for some memory as input and "complete it" to become the memory's full feature vector. For example, a multi-hot input vector which has a '1' for 'wings' and 'yellow' and '0's for every other feature is characteristic of a canary; therefore, we expect that the Modern Hopfield Network retrieves the full feature vector that describes a canary.

2.1 Playing with the Slope Parameter

The first model I implemented is the Modern Hopfield Network with an adjustable slope parameter. The update rule is the same as that of the original Modern Hopfield Network, except that the softmax function incorporates the slope parameter, β (as defined in Section 1.3). For an input feature pattern v, the ith component of v is updated via the rule

$$v_i^{(t+1)} = \sum_{\mu=1}^{N_h} \xi_{i\mu} \text{softmax}_\beta(\sum_{j=1}^{N_f} \xi_{\mu j} v_j^{(t)}).$$

This can more compactly be written in vectorized notation as

$$v^{(t+1)} = W \text{softmax}(\beta W^T v^{(t)}),$$

where W is the matrix of the network's weights, i.e. W lists the feature vectors of each of its stored memories for "pine", "oak", "rose", "daisy", "robin", "canary", "sunfish", and "salmon" as its columns, in that order.

To explore this model's performance, I varied the slope parameter of the net from 0.5 to 10, in steps of 0.5; the number of iterative time steps (called "epochs") of the net was set to 50; and a third variable "num_partial_features" was fixed to 5. For each value of the slope parameter, several incomplete feature patterns (each corresponding to one of the 8 items from the dataset) were passed into the network, and I observed whether the net correctly retrieved the completed feature vector (corresponding to the item being described). Specifically, for each of the 8 items in the dataset, 100 runs of the network were performed; in each run, a sub-pattern of the given item with precisely num_partial_features = 5 randomly selected features (i.e. features that were set to 1) was passed into the Modern Hopfield Network. Subsequently, the accuracies of retrieving each item's full feature vector was calculated. A plot of retrieval accuracy vs. slope parameter is shown in the results section (Figure 2(a)).

2.2 Facilitating New Learning with Automated Slope Parameters

Next, I extended the Modern Hopfield Network to create the "Fluid Hopfield Network," in which there is a dynamically-changing slope parameter for each stored memory; moreover, newly acquired feature patterns (corresponding to newly learned memories) start out with a small slope parameter. The network starts out with a set of stored memories, which are denoted by the column vectors $v_1, \ldots, v_{N_h}$; each memory neuron of the net corresponds to one of these memories. The weights of the network are, as before, stored in the weight matrix $W = (v_1, \ldots, v_{N_h})$. Correspondingly, I initialized a vector of *slope parameters*, $\theta := (\theta_1, \ldots, \theta_{N_h}) = (1, \ldots, 1)$, where θ_j corresponds to the memory vector v_j.

The network runs as follows. When an feature vector $v^{(0)}$ is presented, the net first checks whether this vector is a "sub-pattern" of any of the existing stored memories $v_1, \ldots, v_{N_h}$ – equivalently that $v_j - v^{(0)}$ has no negative entries, for some $j \in \{1, \ldots, N_h\}$. If $v^{(0)}$ is not a "sub-pattern" of any memory, then it gets added to W as a new memory (i.e. we are adding a new neuron to the memory layer of the network!): $W = (v_1, \ldots, v_{N_h}, v^{(0)})$. Crucially, because $v^{(0)}$ is a newly acquired memory, we set its corresponding component in the vector θ to be -1 instead of 1 (like the rest of the "well-grounded" memories): $\theta = (\theta_1, \ldots, \theta_{N_h-1}, \theta_{N_h}) = (\theta_1, \ldots, \theta_{N_h-1}, -1)$, where we have incremented N_h by 1. If $v^{(0)}$ is not a "sub-pattern", we ignore this step and move on.

Next, we run the Fluid Hopfield Network for a preset number of epochs, performing the following update rules, in order:

$$\sigma^{(t)}(\theta^{(t)}) = (\sigma(\theta_1^{(t)}), \ldots, \sigma(\theta_{N_h}^{(t)})) \text{ where } \sigma \text{ denotes the sigmoid function,}$$

$$h_\mu^{(t)} = \sum_{j=1}^{N_f} \xi_{\mu j} v_j^{(t)} \text{ (the hidden memory neuron "activation"),}$$

$$v_j^{(t+1)} = \sum_{\mu=1}^{N_h} \xi_{j\mu} \text{softmax}_{10\sigma^{(t)}(\theta^{(t)})}(h_\mu^{(t)}) = \sum_{\mu=1}^{N_h} \xi_{j\mu} \text{softmax}_{(10\sigma(\theta_1^{(t)}), \ldots, 10\sigma(\theta_{N_h}^{(t)}))}(h_\mu^{(t)}),$$

where we are using the generalized softmax function from Section 1.3. The last update rule is for the individual slope parameters: for each $j \in \{1, \ldots, N_h\}$,

$$\theta_j^{(t+1)} = \theta_j^{(t)} + \frac{v^{(t)} \cdot v_j}{|v^{(t)}||v_j|}\epsilon - d|\theta_j^{(t)}|,$$

where ϵ is a pre-set constant and d is a pre-set "decay" constant (akin to synaptic decay in biological neural networks). This update ensures that memories that are similar (in the cosine similarity metric) to the current input pattern will be given more attention, whereas dissimilar memories will lose attention. After the preset number of "epochs" (iterative cycles), T, of the network, we round $v^{(T)}$ to the closest stored memory vector in the network (Euclidean distance-wise), provided that the closest stored memory vector is already sufficiently close to $v^{(T)}$ - within a pre-specified threshold, such as 0.1. If this is not the case, the network simply returns $v^{(T)}$, and it is assumed that the given feature vector did not converge to a memory (because it was too far from all of the stored memories).

Using the Fluid Hopfield Network, the knowledge acquisition of two novel items was tested: 'lizard' and 'cactus'. Specifically, network parameters of $\epsilon = 0.05$, $d = 0.015$, and num_epochs = 20 were used, and then the feature patterns corresponding to 'lizard' and 'cactus' were presented to the network, one after the other. Subsequently, for each of 10,000 trials, one of the original 8 items, plus 'lizard' and 'cactus', were chosen and a random "sub-pattern" of the given item (consisting of 4 to 6 features) was presented as input to the Fluid Hopfield Network; the cumulative accuracies of pattern completion to the correct memory were then collected over the trials. Additionally, to test how quickly 'lizard' and 'cactus' could be learned in isolation (i.e. no interleaving of other inputs), I generated various plots showing the Euclidean distances of partial 'lizard' and partial 'cactus' representations from each of the stored memories; such plots illustrate how the network adaptively learns to recognize these novel inputs.

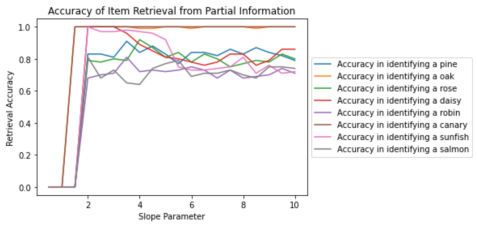
Additionally, as will be further discussed in the results and analysis section, I investigated the effects of adjusting various parameters, such as ϵ, d, and the number of epochs of running the network. In particular, I observed how both anterograde amnesia could "arise" from certain combinations of the ϵ and d parameters, and how the onset of semantic amnesia could be modeled by uniformly driving each memory's slope parameter towards 0.

3 Results and Analysis

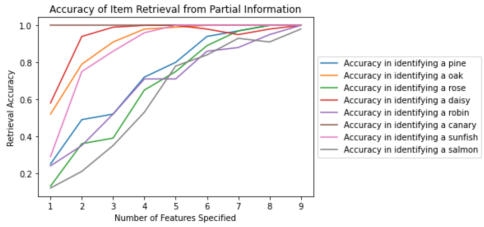
3.1 Exploring the Modern Hopfield Network with a Single Slope Parameter

To get better acquainted with the consequences of adjusting the slope parameter of the Modern Hopfield Network, I generated a plot of slope parameter vs. retrieval accuracy (based on the methodology described in Section 2.1), which is presented in Figure 2(a).

As is evident in Figure 2(a), tuning up the slope parameter generally increases the retrieval accuracies of each item from its partial representations; even just raising it from 1 to 2 leads to retrieval accuracies that are higher than 60% for each item. It should still be noted that tuning up



(a) This figure shows the retrieval accuracies for partial representations (5 randomly chosen features) of each of the 8 original items stored in the network, as the slope parameter is varied.



(b) This figure shows the retrieval accuracies for partial representations (varied from 1 to 9, inclusive) of each of the 8 original items stored in the network, in which the slope parameter is fixed to 3.

Figure 2

the slope parameter even further has diminishing returns on retrieval accuracy; most items have a retrieval accuracy that settles around 80%. This can be explained by the fact that many partial representations of a given item are *also* partial representations of another item. This means that the network may converge to the partial representation of the other item instead; alternatively, the network may also be unable to converge to the feature vector of either item (since the same slope parameter applies to both items) in the case that they equally well could be the completed pattern for the given partial pattern.

In my exploration of the Modern Hopfield Net, I also performed a quick sanity check: the retrieval accuracies of items from their partial representations should increase as more partial features are specified. To this end, I fixed the slope parameter to 3, and varied the number of partial features from 1 to 9, in increments of 1. For each value of 'num_partial_features', I performed 100 runs of the network for each of the 8 items (again) to assess the overall effect of 'num_partial_features' on retrieval accuracy from randomly selected partial patterns with 'num_partial_features' features. The resulting plot is shown in Figure 2(b), and it does indeed show that retrieval accuracy goes to 100% as 'num_partial_features' increases.

3.2 The Fluid Hopfield Network in Action

Now, I will address my findings regarding the Fluid Hopfield Network. As discussed in the Methods section, the first task was to present 'lizard' and 'cactus' to the network and subsequently run 10,000 trials of the network running on randomly interleaved partial feature representations of one of the 10 items (8 original items plus the 2 novel items). The resulting plot of retrieval accuracies is shown in Figure 3.

Figure 3 firstly shows that the Fluid Hopfield Network is indeed capable of consolidating new semantic knowledge, marking an improvement over the ordinary Modern Hopfield Network. However, it also brings to attention an interesting trend: 'cactus' quickly develops a near-1.0 retrieval accuracy, whereas that of the 'lizard' fluctuates before settling around 0.7 – less than the accuracies for the other items. In other words, 'cactus' seems to fit in more with the general schema of memories than 'lizard'. Why is this? In the paper "Hopfield Networks Is All You Need" by

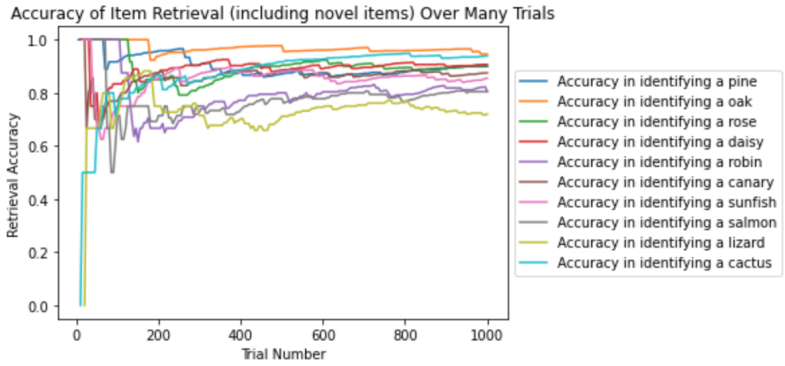


Figure 3. This figure shows the retrieval accuracies for partial representations of each of the 10 items presented to the network. They are all fairly high, though 'lizard' lags behind the others.

Ramsauer et al (2021),[4] the authors introduce a key measure called the *separation* of the memory pattern v_j from the other patterns in the network:

$$\Delta_j = \min_{i \neq j} v_j^T v_j - v_j^T v_i.$$

Comparing 'lizard' to the other patterns, it turns out that 'lizard' has a separation of only 1, whereas 'cactus' has a separation of 2. In other words, the 'cactus' representation has at least 2 ones in places where any other item has 0s. As a result, the network is much more likely to "confuse" a partial representation of 'lizard' with that of a similar item, like 'salmon' or 'sunfish', whereas a partial representation to 'cactus' will typically be more resilient to incorrect pattern completion.

The fact that some items are better consolidated by the network than others brings to light the following question: what if we also treated the 8 original items as "novel items" that are randomly presented to the network rather than already being consolidated in the network upon initialization? To test this idea, I initialized a Fluid Hopfield Net with only one of the 8 original items (randomly chosen); the learning parameters ϵ, d, and num_epochs were as described in the Methods section. Over a series of 100 "exposure trials", the feature vectors corresponding to those of the 10 different items were randomly chosen and passed into the network, just to "acquaint" it with these items. Then, over 1000 trials, randomly chosen partial representations corresponding to one of the 10 items (with 2 to 7 features provided) were passed into the network to test retrieval accuracy. A plot of retrieval accuracy is shown in Figure 4.

Interestingly, this plot is quite similar to that from the previous page, in which the network started out with 8 "well-consolidated" memories. This demonstrates that the Fluid Hopfield Network is capable of consolidating new memories over repeated exposures! As before, retrieval accuracies are high, though 'lizard' still lags behind the other items; the persistence of this symptom suggests that this issue could arise from the fact that 'lizard' only has 7 features, whereas all other items have at least 8 features. Because of this fact, partial representations of 'lizard' may be more likely confounded with other somewhat similar items (like 'sunfish' or 'salmon').

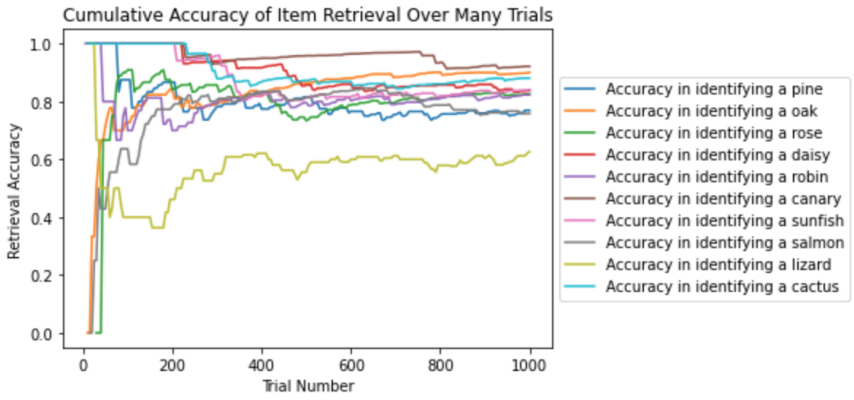
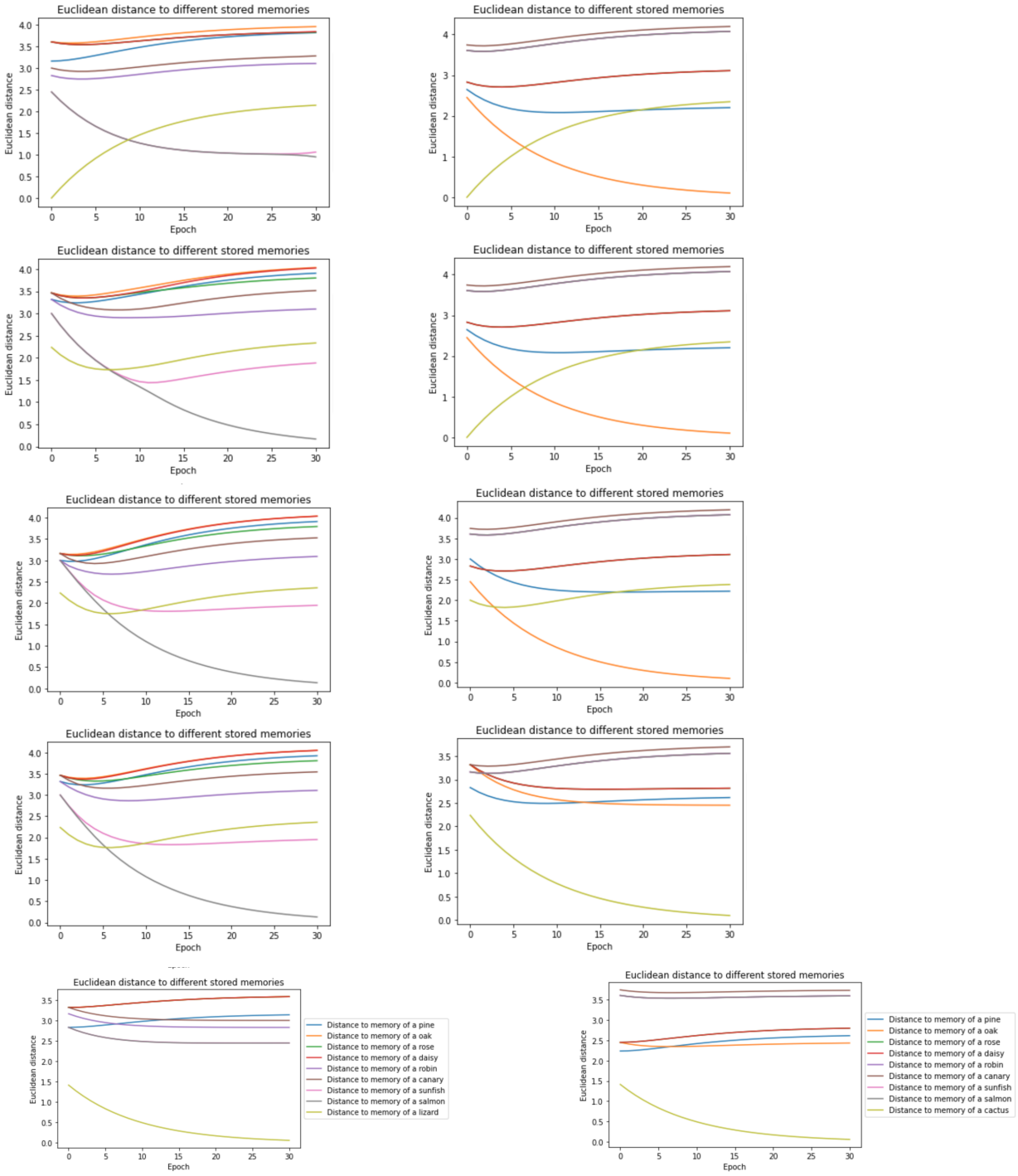


Figure 4. This figure shows the retrieval accuracies of the 10 total items from randomly chosen partial representations (2 to 7 partial features provided), with only one memory initialized into the network to begin with; the other 9 memories are adaptively learned!

As mentioned at the end of the Methods section, I proceeded to investigate the few-shot learning capabilities of the network with regards to the novel items 'lizard' and 'cactus'. Specifically, I decided to create Euclidean-distance plots that would depict the distances from the feature patterns corresponding to 'lizard' and 'cactus' to each of the stored memories of the network, which would change over the epochs of running the network. First, I initialized a Fluid Hopfield Network with the original 8 items as pre-existing memories; all of its learning parameters were as described in the Methods section ($\epsilon = 0.05$, $d = 0.015$, num_epochs = 20), except the time parameter was set to 10 for smoothing the plots in Figure 5. First, the 'lizard' feature vector was passed into the network as an initial "exposure"; unsurprisingly, the network retrieved another item ('salmon'). Then, randomly chosen partial (incomplete) representations of 'lizard' (with 2 to 7 features chosen) were passed into the network until it started retrieving 'lizard' consistently. For each of these trials, a plot of the Euclidean distance from the network's current input representation to each stored item's feature vector (over the 20 epochs of running) was generated. The same process was repeated for 'cactus', except that its randomly chosen partial representations had anywhere from 2 to 8 features, inclusive.

The results are shown in Figure 5, from which it is evident that, within 5 runs of the Fluid Hopfield Network, the network seems to have learned to correctly retrieve 'lizard' and 'cactus' - demonstrating that it is capable of learning in a few-shot manner. One way to interpret the plots is in terms of different memories "competing" with one another to be evoked. In Figure 5(a), for instance, the memory of a 'lizard' is competing with those for 'salmon' and 'sunfish'. While the memory for 'sunfish' initially dominates, the memory for 'lizard' seems to get closer (in the middle 3 plots of Figure 5(a)), before ultimately "winning" in the last plot of Figure 5(a). Similarly, the memory for 'cactus' wins over the memory for 'oak' over the course of 5 runs of the network.



(a) From the top left to the bottom left plot, the network learns to correctly retrieve 'lizard'.

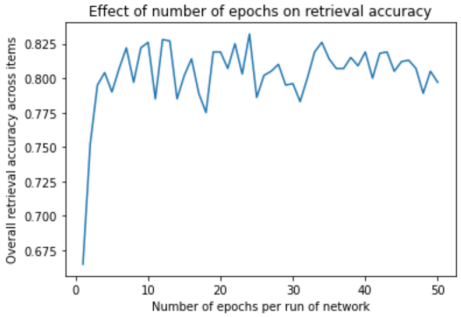
(b) From the top right to the bottom right plot, the network learns to correctly retrieve 'cactus'.

Figure 5. This figure shows the Euclidean distances from the different partial representations of 'lizard' (left half of page) and 'cactus' (right half of page) to each of the stored memories over the epochs. From top to bottom, the plots show how 'lizard' and 'cactus' are learned by the network.

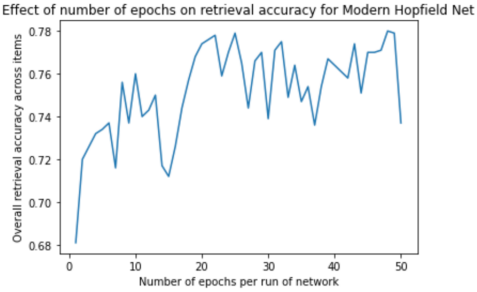
3.3 Exploring Different Combinations of ϵ, d, and num_epochs

Although the Fluid Hopfield Net automates the process of "attention-giving", its performance also depends on some of its hyperparameters: ϵ, d, and num_epochs.

One question about hyperparameter-tuning is the following: how many epochs does the Fluid Hopfield Net need to converge with high accuracy across different input patterns? As it turns out, even 3 epochs is usually enough! To investigate the effect of 'num_epochs' on overall retrieval accuracy, I varied 'num_epochs' from 1 to 50, inclusive; for each value of 'num_epochs', I exposed the Fluid Hopfield Net to 'lizard' and 'cactus' (as usual) and subsequently conducted 1000 trial runs of the Hopfield Net on random partial patterns corresponding to one of the 10 items (2 to 7 features provided). An overall plot of num_epochs versus overall retrieval accuracy (across all the items) is shown in Figure 6(a).



(a) This figure shows the effect of the number of epochs per run of the *Fluid* Hopfield Network on its overall retrieval accuracy across all 10 items.



(b) This figure shows the effect of the number of epochs per run of the *Modern* Hopfield Network on its overall retrieval accuracy across the original 8 items. To compare it with the Fluid Hopfield Net, we set the Modern Hopfield Net's slope parameter to 10 and time parameter to 1.

Figure 6

As is evident from Figure 6(a), retrieval accuracy sharply increases until 3 epochs of running, after which increasing the number of epochs does not have a noticeable effect on retrieval accuracy. This finding suggests that the network converges incredibly rapidly. In turn, this idea suggests that the recurrent collaterals in CA3 of the hippocampus (implicated in pattern completion) may be able to "converge" to the representation of a particular memory within only a few recurrent cycles of neural activity (e.g. 3 cycles).

Additionally, Figure 6(b) shows a graph of number of epochs versus retrieval accuracy across the 8 original items for a *Modern Hopfield Net* with slope parameter 10 and time parameter 1, and with the number of epochs ranging (again) from 1 to 50, inclusive. It serves as a baseline with which to assess the strengths of the Fluid Hopfield Net. In particular, notice the following trends:

1. The Fluid Hopfield Network attains peak performance ($\sim 80\%$) almost immediately, when num_epochs = 3. On the other hand, the Modern Hopfield Network takes tens of epochs to attain a peak retrieval accuracy of about 78%.

2. For num_epochs ≥ 3, the overall retrieval accuracy of the Fluid Hopfield Net consistently stays between 80% to 82%. On the other hand, as seen in Figure 6(b), the Modern Hopfield Network's performance is volatile, fluctuating between 72% and 78%.

All together, we have evidence that the Fluid Hopfield Net approach of "automating attention" via dynamically-updating slope parameters leads to both more accurate *and* more stable retrieval of memories than the Modern Hopfield Net with a single slope parameter (and in fewer epochs too!).

Another important question is how the combination of the ϵ and d parameters in the update rule for θ affect the performance of the network. Observe that, according to the learning rule,

$$\Delta\theta^{(t)} = \frac{v^{(t)} \cdot v_j}{|v^{(t)}||v_j|}\epsilon - d|\theta_j^{(t)}|,$$

which (by design) should be positive if $v^{(t)}$ is sufficiently similar to v_j. In other words, we necessarily need $\epsilon > d$, since if $\frac{v^{(t)} \cdot v_j}{|v^{(t)}||v_j|} = 1$ and $|\theta_j^{(t)}| \geq 1$ (which is a reasonable assumption since each memory vector v_j that is initialized in the network has $\theta_j = 1$), then

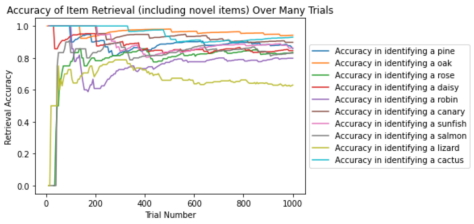
$$\Delta\theta^{(t)} > 0 \iff \frac{v^{(t)} \cdot v_j}{|v^{(t)}||v_j|}\epsilon - d|\theta_j^{(t)}| > 0 \implies \epsilon - d \geq \frac{v^{(t)} \cdot v_j}{|v^{(t)}||v_j|}\epsilon - d|\theta_j^{(t)}| > 0 \implies \epsilon > d.$$

Another guiding rule for hyperparameter tuning can be found by rearranging the learning rule above:

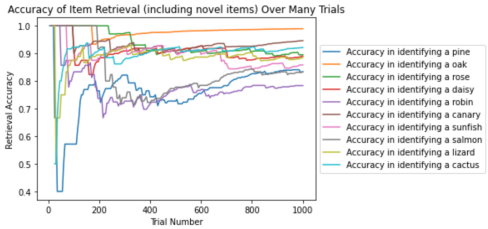
$$|\theta_j^{(t)}| = \frac{\frac{v^{(t)} \cdot v_j}{|v^{(t)}||v_j|}\epsilon - \Delta\theta_j^{(t)}}{d} \leq \frac{\epsilon}{d},$$

which tells us that the slope parameters for each memory are bounded (above and below). This seems desirable, because it prevents any one memory's slope parameter from getting arbitrarily large (in magnitude) compared to the other memories, thereby causing the network to be "biased" towards that memory in a long-term sense. From this result, we make two observations. First, given that it is the *ratio* of ϵ and d that bounds θ_j, we (without loss of generality) fix $\epsilon = 0.5$ and observe what happens when we vary d. Second, given that each θ_j is passed through a sigmoid nonlinearity, we'd like for the interval $I_{\epsilon,d} := [\sigma(\frac{-\epsilon}{d}), \sigma(\frac{\epsilon}{d})]$ to occupy as much of the interval $[0, 1]$ as possible; if $\frac{\epsilon}{d} = 1$, for example, then $I_{\epsilon,d} = [0.27, 0.73]$, whereas for $\frac{\epsilon}{d} = 5$, $I_{\epsilon,d} = [0.007, 0.993]$ is much better at capturing the full range of the sigmoid function. If $I_{\epsilon,d}$ is too narrow - or equivalently, $\frac{\epsilon}{d}$ is too small - then the slope parameters $\{\sigma(\theta_j)\}_{j=1,\ldots,N_h}$ will not be able to vary as much, which would defeat the purpose of having dynamically adjusting slope parameters. In other words, setting $d = 0.1$ is reasonable, because it allows $\frac{\epsilon}{d} = 5$. To see how varying d makes a difference on the network's performance, I ran the Fluid Hopfield Network for only 3 epochs for each of the runsets $(\epsilon, d) = (0.5, 0.1), (0.5, 0.2), (0.5, 0.3), (0.5, 0.4)$, and the resulting retrieval accuracy plots over 1000 randomized trials are shown in Figure 7.

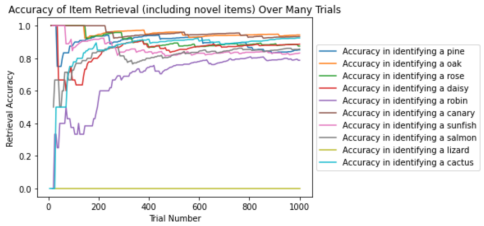
What we observe across the plots is that, as d increases from 0.1, the retrieval accuracies for the



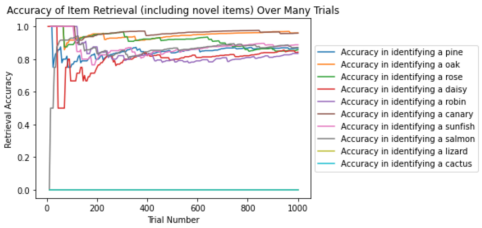
(a) $\epsilon = 0.5, d = 0.1$; retrieval accuracies are fairly high.



(b) $\epsilon = 0.5, d = 0.2$; accuracies remain high overall, though the retrieval accuracy for 'cactus' starts to dip.



(c) $\epsilon = 0.5, d = 0.3$; retrieval accuracy for lizard goes to 0, but remains high for all other items.



(d) $\epsilon = 0.5, d = 0.4$; retrieval accuracies for 'lizard' and 'cactus' are both 0, but remain high for all other items.

Figure 7. Observing the cumulative retrieval accuracies (over 1000 trials) of the 10 memories from randomly selected partial patterns, for different combinations of ϵ and d.

novel items plummets while remaining high for all of the originally consolidated items. In other words, the Fluid Hopfield Network is "developing" *anterograde amnesia*, whereby new memories are incapable of forming, while already-consolidated memories remain intact. In fact, anterograde amnesia can be seen to manifest itself in the learning rule for θ whenver $d \geq \epsilon$. Observe that when the network is exposed to a novel item, v_α, we initialize a new theta weight $\theta_\alpha = -1$ for v_α. Now, for $d \geq \epsilon (= 0.5)$, the learning rule tells us that at timestep $t = 1$,

$$\theta_\alpha^{(1)} \leq \theta_\alpha^{(0)} + \epsilon \frac{v^{(0)} \cdot v_j}{|v^{(0)}||v_j|} - d|\theta_\alpha^{(0)}| \leq -1 + d(\frac{v^{(0)} \cdot v_j}{|v^{(0)}||v_j|} - |-1|) \leq -1 = \theta_\alpha^{(0)}$$

since $\frac{v^{(t)} \cdot v_j}{|v^{(t)}||v_j|} \leq 1$. By the same inductive logic, we see that for any timestep t, $\theta_\alpha^{(t+1)} < \theta_\alpha^{(t)}$, meaning that the attention given to the novel item v_α is never able to increase (and will probably in fact decrease). By the same logic, the network is unable to ever retrieve v_α when a partial pattern of this memory is observed, meaning that v_α never gets consolidated into memory. It should be noted that, whenever $d \geq \epsilon$, θ_j can never increase even for already-consolidated memories j (by the exact same logic as above). The key reason that these memories are retained is because they started out with attention values of $\theta_j^{(0)} = 1 > -1$.

3.4 A Case Study: Semantic Amnesia

Intrigued by the Fluid Hopfield Network's tendency to exhibit anterograde amnesia under certain conditions, one might wonder whether this network can also model *semantic amnesia*, the overall

loss of semantic memory. In this section, I propose that semantic amnesia arises from uniform attentional decay of the slope parameters $\{\sigma(\theta_j)\}_{j=1,\ldots,N_h}$ by a *decay factor*, f, that increments with each overall run of the network. This decay factor presents itself in the generalized softmax function (which is used in the update rule for the network):

$$\text{softmax}_{\sigma/f}(x) = \frac{1}{\sum_{j=1}^{N} \exp(\frac{\sigma_j}{f} x_j)} (\exp(\frac{\sigma_1}{f} x_1), \ldots, \exp(\frac{\sigma_N}{f} x_N))$$

$$\text{where } f = 1 + 0.1 \times (\text{no. of times Hopfield Net has been run}).$$

To visualize the impact of this f parameter on retrieval accuracy, I initialized a Fluid Hopfield Network (whose implementation was slightly modified to include the decay parameter, f) with the hyperparameters $\epsilon = 0.5$, $d = 0.1$, and num_epochs = 20. After exposing the network to 'lizard' and 'cactus' (the novel items), I conducted 1000 trial runs of the network; in each trial run, a partial pattern corresponding to one of the 10 items was randomly chosen and retrieval accuracy was tested. A plot of cumulative retrieval accuracy over the trial runs is shown in Figure 8.

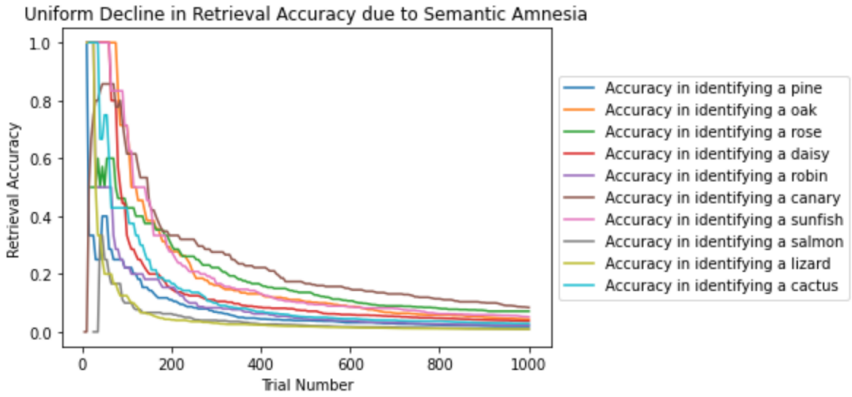
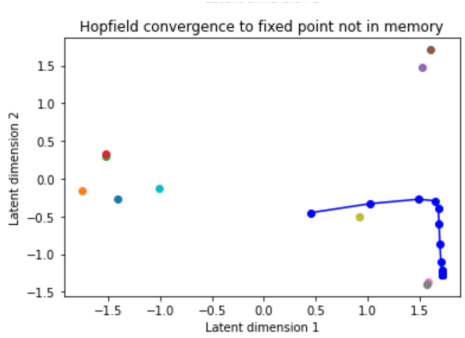


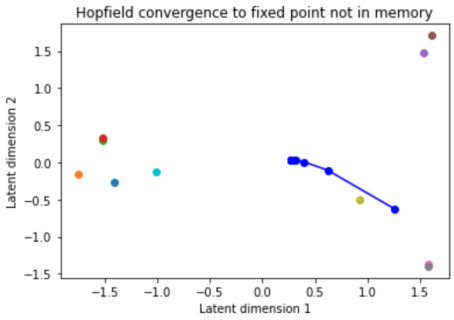
Figure 8. This figure shows the cumulative retrieval accuracies (over 1000 trial runs) of the 10 items from randomly chosen partial representations (2 to 7 partial features provided), in the case where the Fluid Hopfield Network develops semantic amnesia.

From Figure 8, it is evident that, while 'lizard' and 'cactus' are quickly learned by the network, *all* of the items' cumulative retrieval accuracies end up uniformly declining (to 0); this models the onset of semantic amnesia. However, given that each item's retrieval accuracy from a partial pattern of itself is effectively 0, to what vector does this partial pattern *actually* converge?

To analyze the latter question, I performed a principal components analysis of the convergence trajectories of partial patterns that were passed into the network. Specifically, every 50 trials (up until the aforementioned 1000 trials), I zero-centered each of the 10 memory vectors (i.e. shifted their feature vectors the by their mean) and then projected them into two-dimensional space (using the first two principal components, which captured roughly 58% of their variance). Additionally, I shifted the vectors in the convergence trajectory by the mean of the 10 memory vectors and projected



(a) Convergence trajectory on trial run 100 of the network; pattern converges near an existing memory vector.



(b) Convergence trajectory on trial run 550 of the network; pattern converges near the mean of all the memory vectors.

Figure 9

this trajectory into the two-dimensional space as well. Figure 9 shows two sample trajectories (projected in two dimensions) - one on trial 100 and one on trial 550.

On trial run 100, the partial pattern supplied to the network converges near an existing memory (as it normally does); however, by trial run 550 (as well as on later trials), the patterns seem to converge far from any individual memory. Upon a closer look at Figure 9(b), we see that the convergence trajectory ends near $(0, 0)$ in the latent two-dimensional space, nowhere near any of the actual memories themselves. Furthermore, $(0, 0)$ is precisely the projection of the mean of all the memory vectors into this two-dimensional space. Therefore, Figure 9(b) is evidence that, upon the onset of semantic amnesia, partial patterns converge to the "average" of all the stored memories!

To mathematically verify this fact, we have

$$\lim_{f\to\infty} \text{softmax}_{\sigma/f}(x) = \lim_{f\to\infty} \frac{1}{\sum_{j=1}^{N} \exp(\frac{\sigma_j}{f} x_j)} (\exp(\frac{\sigma_1}{f} x_1), \ldots, \exp(\frac{\sigma_N}{f} x_N))$$

$$= \frac{1}{\sum_{j=1}^{N} \exp(0)} (\exp(0), \ldots, \exp(0)) = (\frac{1}{N}, \ldots \frac{1}{N}).$$

Then, the update rule for the network tells us that, for memory j,

$$\lim_{f\to\infty} v_j^{(t+1)} = \lim_{f\to\infty} \sum_{\mu=1}^{N_h} \xi_{j\mu} \text{softmax}_{10\sigma^{(t)}(\theta^{(t)})/f}(h_\mu^{(t)}) = \sum_{\mu=1}^{N_h} \xi_{j\mu} \frac{1}{N}.$$

Therefore, observing that $(\xi_{1\mu}, \xi_{2\mu}, \ldots, \xi_{N_f\mu}) = v_\mu$, the μth stored memory, we find that

$$\lim_{f\to\infty} v^{(t+1)} = \frac{1}{N} \sum_{\mu=1}^{N_h} v_\mu,$$

which is indeed the mean of the memory vectors.

Within the context of this model, semantic amnesia can be thought to arise from a lack of attention. In some sense, the stored memories still exist in the hippocampus: the weight matrix of the network is left intact and it is only the attention values in $\sigma(\theta)$ that are changing (by going to 0). However, by suppressing attention to each memory, individual memories become "silenced", and what is ultimately evoked is a mixture of all of these memories.

4 Discussion

In this paper, I have explored models of the hippocampus that incorporate slope parameters into the Modern Hopfield Network framework. Revisiting the first central question posed in Section 1.1, a Modern Hopfield Network framework - grounded in the idea of pattern completion - is suitable for modeling the association between highly processed hippocampal inputs and their corresponding stored memories. However, to answer the second central question of Section 1.1, the Modern Hopfield Network is not enough. To that end, I have developed an extended model, the Fluid Hopfield Network, that is capable of rapidly acquiring novel semantic information, as discussed in Section 3.2. Moreover, the Fluid Hopfield Net is capable of slightly outperforming the Modern Hopfield Network, retrieval accuracy-wise, after just 3 epochs of running, demonstrating its ability to converge quickly and stably (as seen in Figure 6).

The other big aim of my paper was to explore amnesia in the context of the Fluid Hopfield Network. Given that this network is capable of learning new items, a natural question is whether it is ever possible to prevent this new learning while keeping old memories intact; this is retrograde amnesia, and its occurrence is depicted in Figure 7. Finally, I explored the idea that semantic amnesia emerges from uniform decay in the slope parameters across all the memories of the network, which is discussed in Section 3.4. Interestingly, the conditions that lead to retrograde amnesia - setting $d \geq \epsilon$ - or to semantic amnesia - dividing each $\sigma(\theta_j)$ by the decay parameter f - are formulated rather simply in the framework of my model. This suggests that, when trying to develop a neural network model of a particular brain area, it sometimes helps to come up with a learning rule based on *what might go wrong* in that brain area (e.g. due to a lesion).

While this paper explores various aspects of the Fluid Hopfield Network, many limitations remain and pave the way for future work. One limitation is that applying the sigmoid function to the θ_js is not always ideal. Because the tail ends of this nonlinear function are essentially flat, there is almost no change between $\sigma(3) \approx 0.95$ and $\sigma(4) \approx 0.98$; in other words, as the θ_js increase (as they generally do over many runs of the network), the sigmoid function tends to eliminate much of the difference between different θ_js. As a result, future work might involve experimenting with different nonlinear functions (to be applied to the θ vector) and observing their effects on retrieval accuracy. Furthermore, future work would also involve playing around with different learning rules for θ, guided by different features of the hippocampus. For instance, in the update rule for θ_j, I

might incorporate an exponent into the similarity term:

$$\theta_j^{(t+1)} = \theta_j^{(t)} + \left(\frac{v^{(t)} \cdot v_j}{|v^{(t)}||v_j|} \right)^k \epsilon - d|\theta_j^{(t)}|,$$

where $k > 1$. The idea behind this modification is that a stored memory v_j that is only somewhat similar to the current input pattern $v^{(t)}$ does not merit much more attention, and raising the cosine similarity term to the kth power achieves this effect; only memories v_j that are *really* similar to $v^{(t)}$ warrant more attention. The motivation for implementing this idea is presented in Box 3 of Kumaran et al (2016),[5] whereby pattern separation in the dentate gyrus tends to "amplify" the difference between even seemingly similar patterns.

On a broader note (and to come full circle), it remains to incorporate the Fluid Hopfield Network into the Complementary Learning Systems framework. It would be interesting to see how the Fluid Hopfield Net's attention-based memory retrieval mechanism would interact with a neocortical neural network architecture that simulates some aspect of sensory processing (e.g. vision or language). There is also the monumental question of the Fluid Hopfield Net's biological plausibility. Certainly, one limitation of the Fluid (and Modern) Hopfield Net is the "one memory per neuron" condition for the memory layer. In reality, we expect that memories are distributed across ensembles in the hippocampus. Perhaps more crucial to our discussion, however, is the following question: are dynamically updating "slope parameters" encoded into the computations of hippocampal neurons? It remains to be explored what the specific attentional mechanisms of the hippocampus are.

5 Acknowledgements

I would like to thank Professor Jay McClelland for his guidance regarding focusing my project topic as well as regarding designing experiments that involved playing around with the Modern Hopfield Network (and its variants). Additionally, I am grateful to Andrew Nam for his coding-related advice.

6 Code Availability

For the reader's reference, the Python implementations of my Modern Hopfield Network and Fluid Hopfield Network are available at the following GitHub repository: https://github.com/ShaunakTheHedgehog/fluid-hopfield-network.git.

References

1 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* **102(3)**, 419–457 (1995). https://doi.org/10.1037/0033-295X.102.3.419.

2 J. L. McClelland and T. T. Rogers, "The parallel distributed processing approach to semantic cognition," *Nature Reviews Neuroscience* **4**, 310–322 (2003). https://www.nature.com/articles/nrn1076.

3 D. Krotov and J. J. Hopfield, "Large associative memory problem in neurobiology and machine learning," *ArXiv* **abs/2008.06996** (2021). https://doi.org/10.48550/arXiv.2008.06996.

4 H. Ramsauer, B. Schafl, J. Lehner, *et al.*, "Hopfield networks is all you need," *ArXiv* **abs/2008.02217** (2021). https://doi.org/10.48550/arXiv.2008.02217.

5 D. Kumaran, D. Hassabis, and J. L. McClelland, "What learning systems do intelligent agents need? complementary learning systems theory updated," *Trends in Cognitive Sciences* **20**, 512–534 (2016). https://doi.org/10.1016/j.tics.2016.05.004.