Biological Plausibility in Modern Hopfield Networks

by

Mallory Snow

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Modern Hopfield Networks (HNs) have the ability to store a large number of target memories (e.g. binary patterns) and then recall a memory in its entirety when prompted by a sub-set or perturbed version of it; in this sense, these networks demonstrate properties of a Content Addressable Memory (CAM). Associative memory operates in a similar manner, with memories being elicited by cues of partial or noisy information. Hence, these models provide a basis for modelling associative memory in humans. However, these modern HNs often use functions that rely on many-body synapses and are thus not accurate to neurobiology. More biologically realistic versions of modern HNs have been proposed, although these implementations often still utilize the softmax function. Computing the softmax for a single node in a layer requires the knowledge of all other nodes in that layer, resulting in non-local computations. The softmax function also causes each target memory stored in the network to correspond to the activation of a single hidden node. Such one-hot representations are not robust, and not observed in biological neural networks. In this thesis, we investigate more biologically plausible versions of a modern HN that uses the softmax function. Specifically, we introduce a more biological version of this network by using a local softmax function – the locLSE network. To ensure each target memory has a distributed hidden representation, we propose a network that applies population coding to the hidden layer via the Neural Engineering Framework (NEF) – the Distributed locLSE network. Both proposed networks can learn the connection weights using a local learning rule derived from gradient descent on the energy function. It is found that the proposed, more biologically accurate, versions of the original modern HN still demonstrate capabilities of a CAM at similar rates to the original. Lastly, the NEF is applied to the entire locLSE network by implementing it using a software tool called Nengo. The locLSE network in Nengo is found to behave as a CAM for small datasets of target memories, and provided there are enough neurons in the network.

Acknowledgements

I would like to acknowledge my supervisor, Dr. Jeff Orchard, for introducing me to the field of computational neuroscience and offering endless support and guidance throughout this project. Thank you to Dr. Justin Wan and Dr. Jesse Hoey for being committee members for this thesis. I would also like to thank Terry Stewart, who provided helpful advice for working in Nengo.

Thank you to my family and close friends Michael, Amy, and Jenna for the love and encouragement throughout my Master's.

Dedication

This is dedicated to the my partner, Dylan, for being my biggest supporter and best friend.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Neural Networks (ANNs), inspired by the biological neural networks in the brain, have shown great success in tasks pertaining to a variety of disciplines. Like our brains, ANNs can handle a wide range of complex tasks involving recognition/classification, speech synthesis, image data compression, forecasting and prediction, and so on. However, deep learning algorithms used in ANNs have been criticised for being inconsistent with neurobiology [29].

Nonetheless, there have been many efforts in computational neuroscience to create more biologically realistic ANNs. *Hopfield networks* (HNs), for instance, are Recurrent Neural Networks (RNNs) that can store binary patterns (or "memories") and retrieve them when given a partial or perturbed version of one of these patterns [16]. In this sense, HNs act as a *content-addressable memory* (CAM), which is a system that can retrieve a memory (e.g. a pattern) given sufficient partial or noisy information (part of the memory). For this reason, HNs also provide a model for understanding associative memory in humans. HNs are based on an energy function, E, which combines the cost of neurons being active in the network, and the inconsistency between connected nodes. This consistency is measured by an interaction function, F. Traditional HNs use $F(x) = x^2$ as this interaction function.

Much more recently, Krotov and Hopfield presented *Modern HNs*, or Dense Associative Memory (DAM) models [21]. These models are HNs that use polynomials as the interaction function in the energy. Although this was shown to improve some aspects of these

networks (i.e., their storage capacity, and robustness to adversarial attacks [19]), it moves the implementation of HNs to be further from biology. This is because these models now rely on *many-body synapses*, where the input to a node depends on nonlinear combinations of other nodes. For example, if one uses a quartic interaction function, $F(x) = x^4$, a single synapse would depend on 4 nodes. Since a biological synapse connects only two neurons, these many-body synapses are not biologically plausible. To address this issue, Krotov and Hopfield proposed a model of large associative memory that moves closer to being biologically plausible by coupling a set of N_h *hidden* (or *memory*) nodes (with input currents h_j) to N_v *feature* nodes (with input currents v_i) [20]. However, even in some of Krotov and Hopfield's neurobiological networks, the outputs of nodes still involve "contrastive normalization" via the softmax function. This means that the output of a node is normalized with respect to the currents of all other nodes in the layer, without being connected to them directly. These non-local computations are not biologically plausible. Another factor that is biologically suspicious about this implementation is the fact that each target pattern in the feature nodes corresponds to a one-hot representation in the hidden nodes. The reliance on a one-hot representation is not robust, and not observed in real biology. Hence, non-biological aspects are still present in these modern HNs.

In this thesis, we extend Krotov and Hopfield's recent model of associative memory to make it more consistent with neurobiology. Specifically, to ensure all computations are local, we incorporate a more biological implementation of the softmax function. In addition, we show that this network can *learn* the connection weights using a local learning rule derived from gradient descent on the energy function. We also apply the Neural Engineering Framework (NEF) to the hidden nodes to create a distributed hidden representation rather than a non-biological one-hot hidden representation. We take this a step further by implementing the entire network using a software tool called Nengo which applies the NEF to the entire network. Hence, this thesis works toward a more biologically realistic implementation of a modern HN, and in turn a model of associative memory.

Chapter 2

Background

2.1 Biological Plausibility in Neural Networks

Artificial Intelligence (AI) and neuroscience collectively contribute to our understanding of the brain and motivate the study of biologically plausible neural models. In neuroscience, understanding how external stimuli are represented by the patterns of action potentials they evoke in biological neural networks is still an open question. To answer this question, we need models to interpret experimental neuroscience data [27]. However, although inspired by biology, ANNs are typically inaccurate to how biological neural networks operate. From an AI perspective, we are also motivated to study models that are more biologically accurate as it could overcome certain hurdles and limitations in AI today (the requirement for big data, the presence of adversarial attacks, AI alignment problems, etc).

When it comes to creating more biologically realistic ANNs, there are many things to consider. Namely, to be deemed biologically plausible, an ANN should involve only local computation and local plasticity [4]. Local computation refers to the restriction that a node only performs computations which involve the activity of its input node and their associated weights (not with information from other parts of the network). Local plasticity is the restriction that synaptic changes can only depend on the activity of pre- and post-synaptic nodes. This is one of the reasons that backpropagation, an algorithm used

extensively in practice for training feedforward deep ANNs [31], is not biologically plausible in a lot of implementations. *Hebbian learning* (discussed more in section 2.1.2) is more biological in that it obeys this local plasticity restriction. Another aspect to consider is the architecture of the network. Although there is evidence of roughly hierarchical arrangement and function in the brain [18] – like in the ventral visual pathway – the cortex of the brain is known to be recurrent. It is also theorized that groups (populations/ensembles) of neurons represent external stimuli via non-linearly encoding and linear decoding. This is referred to as population coding (or neural coding) and is the first principle of the Neural Engineering Framework (NEF) [12].

2.1.1 Neural Engineering Framework: Population Coding

Experimental neuroscientists can collect data by directly observing and recording neural activities (firing rate) with respect to environmental stimulus. A group of neurons is said to represent a certain environmental quantity if the each neuron's activity changes as the stimulus does. The first principle of the Neural Engineering Framework (NEF) states that populations (or ensembles) of neurons represent values through a nonlinear encoding and weighted linear decoding.

There are many intermediate steps between the environmental quantity (stimulus) and the neurons that encode it; we can approximate these processes using a linear remapping. If x represents a given external stimulus, we can say that this stimulus is encoded in a distributed representation across N_e neurons. Let $\boldsymbol{a} \in \mathbb{R}^{N_e \times 1}$ represent the activities (or input currents) of these neurons. Using a linear remapping, α, we can approximate the input current to these neurons with respect to the quantity x as,

$$\boldsymbol{a} = \alpha(x) = \boldsymbol{W}_e x + \boldsymbol{b}_e. \tag{2.1}$$

Here, $\boldsymbol{W}_e \in \mathbb{R}^{N_e \times 1}$ are the encoding weights which control the change in slope and each neuron's preferred direction for this stimulus, while $\boldsymbol{b}_e \in \mathbb{R}^{N_e \times 1}$ are the encoding biases.

We can then describe the firing rate of these neurons with respect to their input currents using a non-linear encoding, $G(\boldsymbol{a})$, as the NEF suggests. This can be done using common

non-linear functions like $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ or $\sigma(x) = \frac{1}{1+e^{-x}}$, but we can also use more biologically realistic functions. For instance, the firing rate of neuron i according to the classic Leaky Integrate-and-Fire (LIF) neuron model is given by,

$$G(\boldsymbol{a}_i) = \begin{cases} \left[\tau_{\text{ref}} - \tau_m \log\left(1 - \frac{a_{\text{th}}}{\boldsymbol{a}_i}\right)\right]^{-1} & \text{for } \boldsymbol{a}_i > a_{\text{th}} \\ 0 & \text{else} \end{cases}, \tag{2.2}$$

where τ_{ref} is the refractory period of the neuron, τ_m is the membrane time constant of the neuron, and a_{th} is the threshold of the neuron. The LIF neuron model describes the sub-threshold membrane potential, or voltage, of a neuron. It does not model the neuron's spikes themselves, but rather simply records when a spike occurs. After the neuron spikes, it remains dormant during its refractory period, τ_{ref}, after which it starts integrating again.

It is important to keep in mind that in general, x is not just a single valued stimulus, but rather we can have multiple stimuli that we want to encode in a population of neurons, and each stimulus can be dependent on multiple different variables. Moreover, we could have a matrix, $\boldsymbol{X} \in \mathbb{R}^{D \times M}$, containing M stimuli, each of which is dependent on D variables. Likewise, the ensemble of N_e neurons encode these M stimuli differently; thus, we can have a matrix $\boldsymbol{A} \in \mathbb{R}^{N_e \times M}$ which stores the activities of the N_e neurons for each of the M stimuli via equation (2.1) where we now have $\boldsymbol{W}_e \in \mathbb{R}^{N_e \times D}$.

We can then linearly decode the stimulus, $\hat{\boldsymbol{X}}$, from the neural activities, $\boldsymbol{A}$, using decoding weights $\boldsymbol{D}_X \in \mathbb{R}^{D \times N_e}$, such that $\hat{\boldsymbol{X}} = \boldsymbol{D}_X \boldsymbol{A}$. To do this, we need to solve for the decoding weights that minimize the error, Er, between the actual stimulus values and the decoded values,

$$\begin{aligned} \text{argmin}_{\boldsymbol{D}_X}\ Er &= \text{argmin}_{\boldsymbol{D}_X} \|\boldsymbol{X} - \hat{\boldsymbol{X}}\| && (2.3) \\ &= \text{argmin}_{\boldsymbol{D}_X} \|\boldsymbol{X} - \boldsymbol{D}_X \boldsymbol{A}\|. && (2.4) \end{aligned}$$

To solve for the decoding weights, we create samples to "train" the decoding weights with. To this end, we first create a set of N_s samples of our stimuli, $\boldsymbol{X}_s \in \mathbb{R}^{D \times N_s}$. Next, we can compute the corresponding activities for these samples, $\boldsymbol{A}_s$, and the approximate decoded samples, $\hat{\boldsymbol{X}}_s = \boldsymbol{A}_s \boldsymbol{D}_X$. Minimizing the difference between the actual and decoded

sample values with respect to $\boldsymbol{D}_X$ can be done using the Least-squares Minimization solution,

$$\boldsymbol{D}_X = \left(\boldsymbol{A}_s \boldsymbol{A}_s^T\right)^{-1} \boldsymbol{A}_s \boldsymbol{X}_s. \tag{2.5}$$

Hence, as per the first principle of the NEF, we have that an ensemble of neurons represents external stimuli via a nonlinear encoding and linear decoding,

$$\begin{aligned} \boldsymbol{A} &= G[\boldsymbol{W}_e \boldsymbol{X} + \boldsymbol{b}_e] && \text{encoding,} \\ \hat{\boldsymbol{X}} &= \boldsymbol{D}_X \boldsymbol{A} && \text{decoding.} \end{aligned}$$

2.1.2 Hebbian Learning

Hebbian learning is based on Hebb's theory that synaptic plasticity is dependent on the correlated activity of pre- and post- synaptic neurons [15]. That is, the synaptic connection between two neurons increases in strength when the pre-synaptic neuron repeatedly or persistently takes part in the activity of the post-synaptic neuron. This theory is often shortened to the phrase *"neurons that fire together wire together"*. In terms of ANNs this simply translates to increased connection weights between two neurons if they are simultaneously active. Mathematically this can be described as

$$w_{ij} = x_i x_j, \tag{2.6}$$

where x_i and x_j represent the activities of nodes i and j respectively for a given stimulus, and w_{ij} is the connection weight from nodes j to i. So, two nodes that have the same sign will have strong positive weights while opposite signs lead to negative weights. Moreover, the weights can be thought to represent the co-activation of the nodes in the network for a set of external stimuli. When we want to store several stimuli in network, we can set the weights to be the average co-activation over all the stimuli. Since this learning rule clearly obeys local plasticity, it is biologically plausible by our previous definition.

Hebb's theory describes an engram, meaning it details the means by which memories can be stored in response to stimuli. One can imagine that if a given external stimulus repeatedly causes the same pattern of neural activity then this set of active neurons will become increasingly inter-associated. This idea that synaptic plasticity strengthens (or

decreases) with the simultaneous activation (or inactivation) of neurons is also referred to as Long-Term Potentiation (or Long-Term Depression), i.e. LTP (or LTD). These mechanisms are often used to explain associative learning and memory [36, 40]. Sets of neurons can become so strongly associated for given memories that when they are prompted by a partial or perturbed version of one of these memories, the neurons automatically respond as a whole, reconstructing the entire memory.

2.2 Hopfield Networks

John Hopfield published an influential paper dating back to 1982 in which he introduced a method for using RNNs as a content-addressable memory system (CAM). These Hopfield Networks (HNs) store binary patterns (i.e. target memories) and can then retrieve them by converging to the closest pattern when given a partial or perturbed version [16]. These networks were introduced as being fully connected RNNs. If we have a HN consisting of N nodes, each node, $\boldsymbol{v}_i$, is set to be ± 1 since the learned patterns are binary strings. A visual of such a network is illustrated in Figure 2.1.

Let $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$ represent the set of target memories (binary patterns) that we want to store in the HN, i.e. the *memory dataset*. Then, $\boldsymbol{X}$ is a $D \times M$ matrix where M (indexed by m) is the number of targets and D (indexed by d) is the number of bits in each pattern, with each $\boldsymbol{X}_{dm} \in \{-1, 1\}$. The synaptic connections in a HN follow Hebbian learning and thus the weight matrix, $\boldsymbol{W}$, is set to be $\boldsymbol{W} = \frac{1}{M}\boldsymbol{X}\boldsymbol{X}^T$. Hence, the synaptic weight connecting nodes $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, $\boldsymbol{W}_{ij}$, will become increasingly positive as the set of memory patterns for these nodes become more highly correlated (i.e., the more the nodes share the same sign).

Given a vector, $\boldsymbol{v}$ representing the state of the nodes in the network, each node, $\boldsymbol{v}_i$ then changes/updates its value according to the following update rule,

$$\boldsymbol{v}_i = \begin{cases} 1 & \text{if } \sum_{j \neq i} \boldsymbol{W}_{ij}\boldsymbol{v}_j \geq 0, \\ -1 & \text{if } \sum_{j \neq i} \boldsymbol{W}_{ij}\boldsymbol{v}_j < 0. \end{cases} \tag{2.7}$$

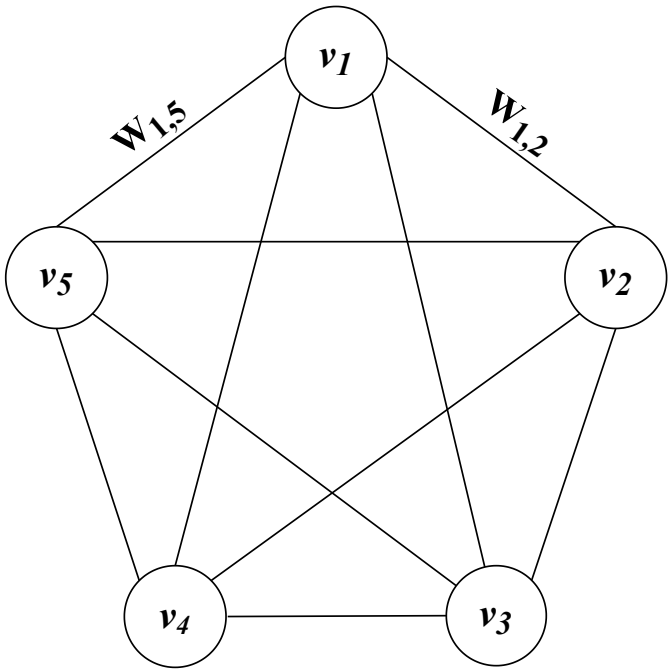


Figure 2.1: A diagram of a traditional HN with $N = 5$ nodes.

Hopfield recognized the connection between these models and the Ising model in physics. The Ising model describes a lattice of interacting magnetic dipoles, where, similar to HNs, each dipole can be "up" or "down" (± 1), and this state is dependant on its neighbours [16]. Like in the Ising model, one can write an energy function, E, describing the system. In terms of HNs, E can be thought of as combining the "cost" of nodes being active in the network by measuring the inconsistency/conflict between connected nodes. This consistency between adjacent nodes is measured using an interaction function, F, which, in traditional HNs, is defined as $F(x) = x^2$. Hence, the energy function for the traditional

HN is written as follows,

$$\begin{aligned} E &= -\frac{1}{2}\sum_i \sum_{j\neq i} \boldsymbol{W}_{ij}\boldsymbol{v}_i\boldsymbol{v}_j \\ &= -\frac{1}{2}\boldsymbol{v}^T\boldsymbol{W}\boldsymbol{v} \\ &= -\frac{1}{2}\sum_m^M (\boldsymbol{X}^m\boldsymbol{v})^2, \text{ where } \boldsymbol{X}^m \text{ is the } m^{th} \text{ target pattern,} \\ &= -\frac{1}{2}\sum_m^M F(\boldsymbol{X}^m\boldsymbol{v}), \end{aligned} \tag{2.8}$$

where again, $F(x) = x^2$. It is also important to point out that the update rule for the nodes in equation (2.8) can be derived by performing gradient decent on E. That is, the algorithm for updating $\boldsymbol{v}$ causes E to monotonically decrease, and the states continue to change until a local minimum is reached. This strategy drives a lot of theory behind unsupervised learning; since we don't necessarily know the target output, rather than optimizing a loss function, we use the energy which gives us something to minimize.

2.2.1 Modern Hopfield Networks

Much more recently, Krotov and Hopfield presented modern HNs, or Dense Associative Memory (DAM) models, which are HNs that use interaction functions of the form $F(x) = x^q, q \in \mathbb{Z}^+$ in the energy [21]. Clearly, the case of $q = 2$ reduces to the traditional HN formulation. Modern HNs were shown to have an increased storage capacity when compared to traditional HNs. Specifically, for a network of N nodes, the original HN can recall approximately $0.138N$ different patterns whereas the modern HN increased this capacity to N^{q-1}. It was later shown that modern HNs also improved the network's robustness to adversarial input [19].

2.2.2 Hopfield Networks and Biology

HNs have also gained interest for their link to human memory. These networks act as a model of *auto-associative memory* since they can recover items from memory that have

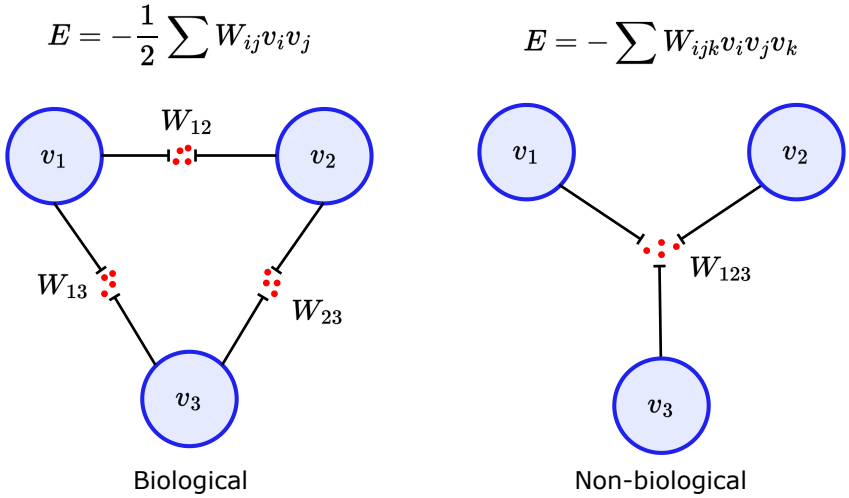


Figure 2.2: A diagram of traditional HN (left) and modern HN with interaction function $F(x) = x^3$ (right). In this case, the modern HN has weights that are dependent on three nodes which is biologically implausible. Image reproduced from [20].

been distorted [8]. This can also be thought of as a form of *cued-recall*, where an entire memory is recalled when prompted by a cue [10, 39]. For this reason, implementing HNs in a biologically plausible manner moves us closer to a model that explains the process the brain implements in certain memory tasks such as auto-associative memory and pattern completion.

Although modern HNs improved some aspects of HNs, it moved the implementation further from being biologically plausible. This is due to the many-body synapses introduced from the polynomial interaction function. In Figure 2.2 we see the traditional HN on the left and a modern HN on the right with a cubic interaction function, $F(x) = x^3$. We can see that the weights here depend on three nodes, and since a biological synapse can only connect two nodes, these many-body synapses are biologically implausible.

To address this many-body synapse issue, Krotov and Hopfield proposed a model of large associative memory that moves closer to being biologically plausible [20]. Their approach was to couple a set of N_h hidden nodes, $\boldsymbol{h}$ (indexed by j), to a set of N_v feature nodes, $\boldsymbol{v}$ (indexed by i). The full network of $N_v + N_h$ nodes forms a bipartite graph, which

behaves like a classifier; each target pattern in the feature nodes corresponds to a one-hot setting of the hidden nodes, and *vice versa*.

Again, let $\boldsymbol{X} \in \mathbb{R}^{D\times M}$ be our memory dataset which consists of M targets, each with D bits. In these networks, $\boldsymbol{\xi} \in \mathbb{R}^{D\times M}$ is the connection-weight matrix in which $\boldsymbol{\xi}_{ij}$ represents the strength of the synapse from hidden node j to the feature node i. The target memories are stored in the connection weights, i.e., $\boldsymbol{X} = \boldsymbol{\xi}$, which connect the hidden nodes to the feature nodes. The connections going in the other direction are symmetric to these weights. For example, let $\bar{\boldsymbol{\xi}} = \boldsymbol{\xi}^T$, we have that $\boldsymbol{\xi}_{ij} = \bar{\boldsymbol{\xi}}_{ji}$. In other words, the connection from h_j to v_i, $\boldsymbol{\xi}_{ij}$, has the same weight as the connection from v_i to h_j, $\bar{\boldsymbol{\xi}}_{ji} = (\boldsymbol{\xi}^T)_{ji}$. There are no synaptic connections among the hidden nodes, or feature nodes. The outputs of the feature and memory nodes are denoted by $\boldsymbol{g}_i$ and $\boldsymbol{p}_j$ respectively, which are (non-linear) functions of their corresponding input currents. These functions are chosen such that $\boldsymbol{p}_j = \frac{\partial L_{\boldsymbol{h}}}{\partial \boldsymbol{h}_j}$ and $\boldsymbol{g}_i = \frac{\partial L_{\boldsymbol{v}}}{\partial \boldsymbol{v}_i}$, for some Lagrange functions $L_{\boldsymbol{h}}$ and $L_{\boldsymbol{v}}$. The time constants for the feature and memory nodes are denoted by τ_v and τ_h respectively. Hence, the entire network of $N_v + N_h$ nodes can be described in continuous time by the following dynamical system,

$$\tau_v \frac{d\boldsymbol{v}}{dt} = \boldsymbol{\xi}\boldsymbol{p} - \boldsymbol{v} + \boldsymbol{I} \ , \tag{2.9}$$

$$\tau_h \frac{d\boldsymbol{h}}{dt} = \boldsymbol{\xi}^T\boldsymbol{g} - \boldsymbol{h} \ , \tag{2.10}$$

where $\boldsymbol{I}$ denotes the input current to feature nodes $\boldsymbol{v}$. A small example of the network discussed above can be seen in Figure 2.3.

We can see that the temporal updates for the nodes in a given group are determined by the inputs from other nodes and their own state. Hence, the energy of these networks can be written as the sum of three terms: one term for the feature nodes, one term for the hidden nodes, and one term for the interaction between the two groups of nodes,

$$E = \left[\sum_{i=1}^{N_v} (\boldsymbol{v}_i - \boldsymbol{I}_i)\boldsymbol{g}_i - L_{\boldsymbol{v}}\right] + \left[\sum_{j=1}^{N_h} \boldsymbol{h}_j\boldsymbol{p}_j - L_{\boldsymbol{h}}\right] - \sum_{i,j} \boldsymbol{g}_i\boldsymbol{\xi}_{ij}\boldsymbol{p}_j \ . \tag{2.11}$$

This energy function decreases on the dynamical trajectory of the nodes. That is, the changes in $\boldsymbol{v}$ and $\boldsymbol{h}$ only cause the energy to decrease until reaching a local minimum which correspond to the equilibrium states of the network.

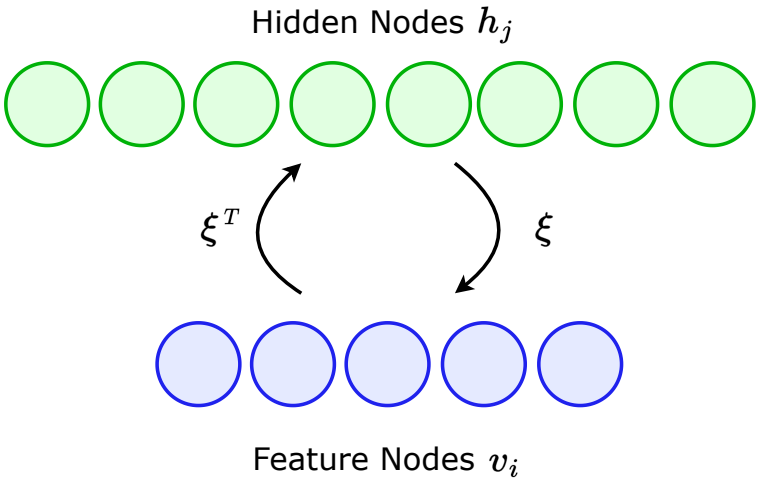


Figure 2.3: A diagram of Krotov and Hopfield's associative memory network from [20]. This example has $N_v = 5$ feature nodes, $N_h = 8$ hidden nodes, and symmetric connection weights, $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^T$. Image reproduced from [20].

Even in some of Krotov and Hopfield's neurobiological networks, the outputs of the feature and memory nodes still involve "contrastive normalization". This refers to the normalization of each node's activity with respect to the activities of all other nodes in the layer [20]. The model focused on in this work corresponds to *Model B* in [20], however we refer to it as the *LSE network*.

2.2.3 The LSE Network

The modern HN studied here, the LSE network, involves outputs (activation functions) of the form $\boldsymbol{g}_i = \boldsymbol{v}_i$, and

$$\boldsymbol{p}_j = \mathcal{S}(\boldsymbol{h})_j = [\text{softmax}(\boldsymbol{h})]_j = \frac{e^{\boldsymbol{h}_j}}{\sum_k e^{\boldsymbol{h}_k}} \ . \tag{2.12}$$

Substituting $\boldsymbol{p} = \mathcal{S}(\boldsymbol{h})$ and $\boldsymbol{g} = \boldsymbol{v}$ into equations (2.9) and (2.10) gives the following dynamics,

$$\tau_v \frac{d\boldsymbol{v}}{dt} = (1-\beta)\boldsymbol{\xi}\mathcal{S}(\boldsymbol{h}) - \boldsymbol{v} + \beta\boldsymbol{I}, \tag{2.13}$$

$$\tau_h \frac{d\boldsymbol{h}}{dt} = \boldsymbol{\xi}^T\boldsymbol{v} - \boldsymbol{h}. \tag{2.14}$$

In equation (2.13), β is introduced as a weighting factor which allows the input to be turned on, off, or decayed. The input to the feature nodes, $\boldsymbol{I}$, can be 'turned off' by setting $\beta = 0$. The input can be 'turned on' by setting $\beta = 1$. The input can be decayed to 0 (off) by setting $\beta = e^{-bt}$, where b describes the rate of this decay.

Note that the LSE function is defined as $\text{LSE}(\boldsymbol{h}) = \log\left(\sum_k e^{\boldsymbol{h}_k}\right)$, the gradient of which is equal to the softmax function, $\frac{\partial}{\partial \boldsymbol{h}_j}(\text{LSE}(\boldsymbol{h})) = \mathcal{S}(\boldsymbol{h})_j$. Also note the following relationship of $\frac{\partial}{\partial \boldsymbol{v}_i}\left(\frac{1}{2}\sum_i \boldsymbol{v}_i^2\right) = \boldsymbol{v}_i$. Hence, the Lagrange functions here are $L_{\boldsymbol{h}} = \log\left(\sum_j e^{\boldsymbol{h}_j}\right) = \text{LSE}(\boldsymbol{h})$, and $L_{\boldsymbol{v}} = \frac{1}{2}\sum_i \boldsymbol{v}_i^2$. Applying this to equation (2.11) yields the following energy function,

$$E(t) = \sum_{i=1}^{N_v}\left(\frac{1}{2}\boldsymbol{v}_i^2 - \boldsymbol{v}_i\boldsymbol{I}_i\right) + \sum_{j=1}^{N_h}\left(\boldsymbol{h}_j\,\mathcal{S}(\boldsymbol{h})_j - \text{LSE}(\boldsymbol{h})\right) - \sum_{j,i}\boldsymbol{v}_i\boldsymbol{\xi}_{ij}\mathcal{S}(\boldsymbol{h})_j. \tag{2.15}$$

Again, the nodes in the network change dynamically to minimize the energy of the network.

As mentioned in the previous chapter, biological implausibility here is due to the use of softmax, $\mathcal{S}(\boldsymbol{h})$, in equation (2.13). The output of hidden node j will depend on **all** other hidden nodes, even though they are not connected. This non-local computation poses a degree of biological implausibility. It is also biologically suspect that the hidden nodes ideally represent the target patterns using a one-hot representation. Both of these aspects are dealt with in this thesis.

Chapter 3

Local LSE Network

Here, we present a more biologically plausible version of the LSE network that only uses local computations - the *locLSE network*. This network formed the basis for our work that was published at a cognitive science conference [34].

Recall that to be biologically plausible, the output of each node must only depend on its own input, i.e. involve local computation. Computing $\mathcal{S}(\boldsymbol{h})_j$, i.e. the softmax of hidden node j, involves normalization with respect to all other hidden node activities, and is thus a non-local computation. Hence, $\mathcal{S}(\boldsymbol{h})_j$ cannot be computed by a single node alone ($\boldsymbol{h}_j$) – it must be implemented by a *network* of nodes. Thus, we first outline the *local softmax network*, which, as the name suggests, computes the softmax function using only local computations.

3.1 Local Softmax Network

To build the local softmax network, we first consider $\mathcal{S}(\boldsymbol{h})_j$ in the log domain:

$$\begin{aligned}\log\left(\mathcal{S}(\boldsymbol{h})_j\right) &= \log\left(\frac{e^{\boldsymbol{h}_j}}{\sum_k^{N_h} e^{\boldsymbol{h}_k}}\right) \\ &= \log\left(e^{\boldsymbol{h}_j}\right) - \log\left(\sum_k^{N_h} e^{\boldsymbol{h}_k}\right) \\ &= \boldsymbol{h}_j - \mathrm{LSE}(\boldsymbol{h}).\end{aligned}$$

Some basic log rules are applied here in order to write $\log\left(\mathcal{S}(\boldsymbol{h})_j\right)$ as the subtraction of two terms. Suppose we have a node, c, that stores $\mathrm{LSE}(\boldsymbol{h}) = \log\left(\sum_k^{N_h} e^{\boldsymbol{h}_k}\right)$. Then,

$$\log\left(\mathcal{S}(\boldsymbol{h})_j\right) = \boldsymbol{h}_j - c\ .$$

Then, if we define a set of nodes $\boldsymbol{f}$ such that $\boldsymbol{f}_j = \boldsymbol{h}_j - c$, we get $\log\left(\mathcal{S}(\boldsymbol{h})_j\right) = \boldsymbol{f}_j$. Finally, we can store $\mathcal{S}(\boldsymbol{h})_j$ in node $\boldsymbol{p}_j$ by defining $\boldsymbol{p}_j = e^{\boldsymbol{f}_j}$,

$$\mathcal{S}(\boldsymbol{h})_j = e^{\boldsymbol{f}_j} = \boldsymbol{p}_j\ .$$

This is similar to Bogacz and Gurney's method of using logarithms to do normalization of probabilities [5].

We can model this local softmax network as a system of differential equations,

$$\begin{aligned}\tau_s \frac{dc}{dt} &= \log\left(\sum_{j=1}^{N_h} e^{\boldsymbol{h}_j}\right) - c \\ \tau_s \frac{d\boldsymbol{f}}{dt} &= \boldsymbol{h} - c - \boldsymbol{f} \\ \tau_s \frac{d\boldsymbol{p}}{dt} &= e^{\boldsymbol{f}} - \boldsymbol{p}\,,\end{aligned}$$

where τ_s is the time constant for computing the softmax. For a fixed $\boldsymbol{h}$, the equilibrium solution of this dynamical system clearly corresponds to $\boldsymbol{p}$ holding the softmax of $\boldsymbol{h}$ (i.e., $\boldsymbol{p}_j = \mathcal{S}(\boldsymbol{h})_j$). This local softmax network is illustrated in Figure 3.1. To verify that the

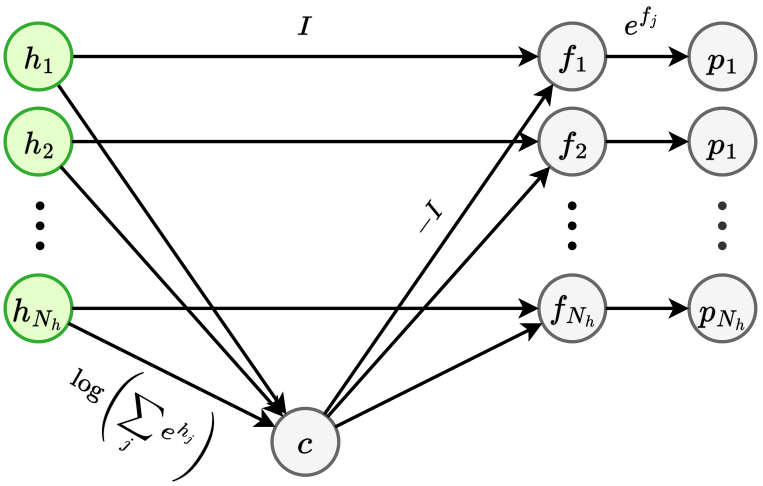


Figure 3.1: A diagram of the local softmax subnetwork. The $\boldsymbol{p}$ nodes store the softmax of the $\boldsymbol{h}$ nodes, $\boldsymbol{p}_j = \mathcal{S}(\boldsymbol{h})_j$.

local softmax network does indeed compute the softmax of a given vector, $\boldsymbol{h}$, we ran some experiments in which $\boldsymbol{h}$ is held constant and the rest of the network is run to equilibrium. To solve this system of equations, we used the `solve_ivp` function from the SciPy python library. Within this function, the Runge-Kutta integration method is used by default to solve the system of differential equations. It is clear from Figure 3.2 that the $\boldsymbol{p}_j$ nodes converge to the expected values, corresponding to the softmax of the hidden nodes.

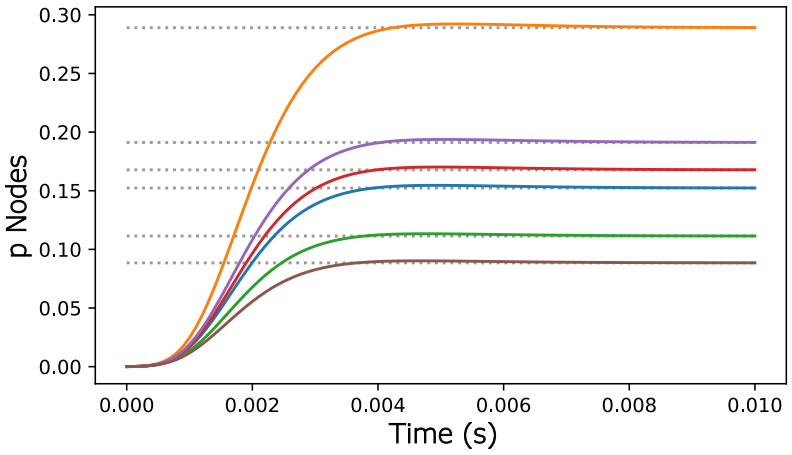
After verifying that the local softmax network can successfully compute the softmax function, we incorporate it into the LSE network by replacing $\mathcal{S}(\boldsymbol{h})$ in equation (2.13) with $e^{\boldsymbol{f}}$. This results in the following set of coupled differential equations,

$$\tau_v \frac{d\boldsymbol{v}}{dt} = (1-\beta)\boldsymbol{\xi} e^{\boldsymbol{f}} - \boldsymbol{v} + \beta \boldsymbol{I} \tag{3.1}$$

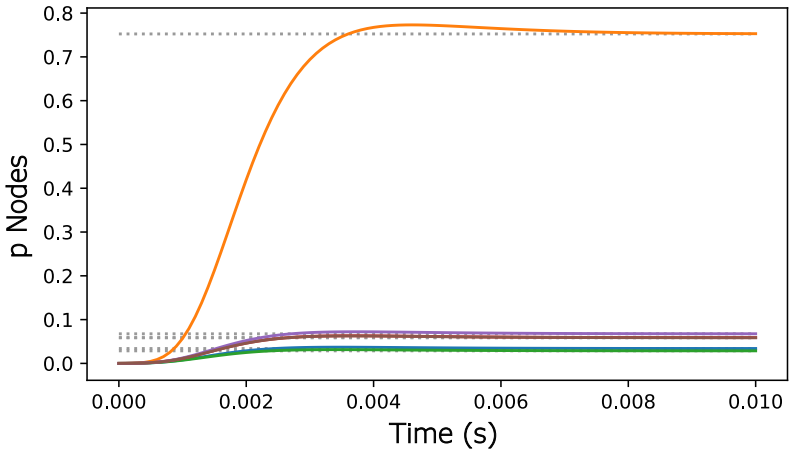
$$\tau_h \frac{d\boldsymbol{h}}{dt} = \boldsymbol{\xi}^T \boldsymbol{v} - \boldsymbol{h} \tag{3.2}$$

$$\tau_s \frac{dc}{dt} = \log\left(\sum_{j=1}^{N_h} e^{\boldsymbol{h}_j}\right) - c \tag{3.3}$$

$$\tau_s \frac{d\boldsymbol{f}}{dt} = \boldsymbol{h} - c - \boldsymbol{f}. \tag{3.4}$$



(a) Random $\boldsymbol{h}$



(b) One node in $\boldsymbol{h}$ is larger than the others

Figure 3.2: The dynamics of the local softmax network when $\boldsymbol{h}$ is held constant (a) at random values, and (b) at random values with one node larger than the rest. The plots show the evolution of the $\boldsymbol{p}_j$ nodes, as shown in Fig. 3.1. The solid lines represent the activities of the $\boldsymbol{p}_j$ nodes, with each colour representing a different node. The horizontal dotted lines show the true values of $\mathcal{S}(\boldsymbol{h})$. Note that all the components of h sum to 1.

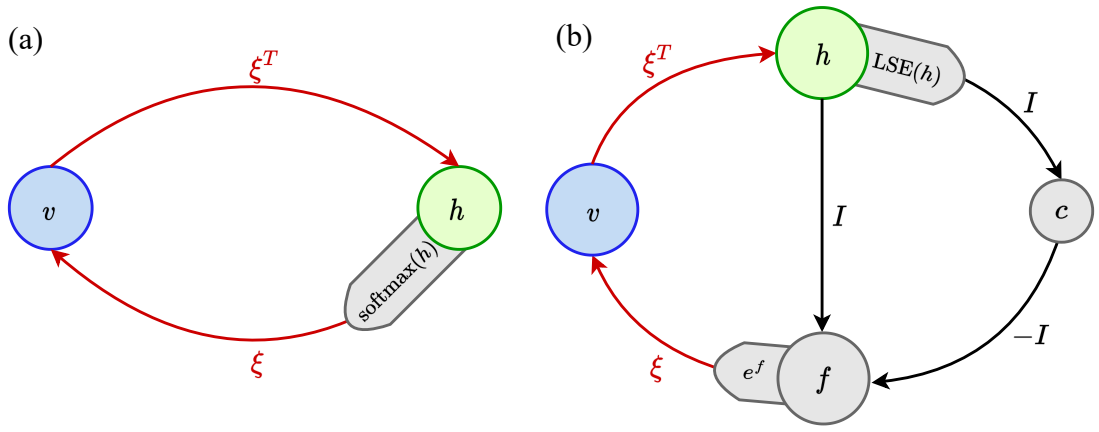


Figure 3.3: A diagram of (a) the LSE network from [20] and (b) the locLSE network. Each circle can represent a population of nodes. Specifically, the blue $\boldsymbol{v}$ circle represents a population of N_v nodes, while $\boldsymbol{h}$ and $\boldsymbol{f}$ each represent a population of N_h nodes. Here, I represents an identity mapping (connection weights equal to the identity).

The $\boldsymbol{p}$ group is no longer needed here as we can directly project the exponential of $\boldsymbol{f}$ to $\boldsymbol{v}$. It is important to note that we need the softmax sub-network to produce its output quickly so that the softmax function appears instantaneous relative to the rest of the network. Thus, we require that the time constant for computing the softmax function, τ_s, is smaller than τ_v and τ_h. An outline of the LSE network before and after the addition of the local softmax subnetwork is depicted in Figure 3.3 (a) and (b).

3.2 Learning Connection Weights

Rather than storing the memory dataset in the network by directly setting them as the weights, a more biological approach would involve *learning* these weights in order to store the patterns in the network. To do this, we hold the feature and hidden nodes constant and consider how we might change $\boldsymbol{\xi}$ to minimize the energy and learn a set of target patterns.

In equation (2.15), we see that these weights only appear in the last term, such that

$$\frac{\partial E}{\partial \boldsymbol{\xi}_{ij}} = -\boldsymbol{v}_i \mathcal{S}(\boldsymbol{h})_j, \tag{3.5}$$

where $\boldsymbol{v}$ and $\boldsymbol{h}$ are clamped to their ideal values for a single target. We can learn the weights for each target "one at a time" by clamping the feature and hidden nodes to their ideal values for each target memory and cycling through them all. Or, we can implement this batch-wise using matrices, so that the feature and hidden nodes are clamped to their ideal values for the entire memory dataset, thus learning the weights for all targets in one go.

Specifically, consider a memory dataset, $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$, of M target memories with D bits each. We set the feature and hidden nodes to their ideal target values for this entire dataset, $\boldsymbol{v} = \boldsymbol{X}$, $\boldsymbol{h} = \boldsymbol{X}^T \boldsymbol{v}$, and clamp the derivatives in equations (3.1) and (3.2) to zero so they do not change. Then, we can initialize the weight matrix $\boldsymbol{\xi}$ randomly near zero, and adjust them to minimize the energy. Hence, writing equation (3.5) batch-wise gives,

$$\boldsymbol{\nabla}_{\boldsymbol{\xi}} \boldsymbol{E} = -\boldsymbol{v}\ \mathcal{S}(\boldsymbol{h})\ .$$

Again, $\boldsymbol{v}$ here contains all the targets (column-wise) that we wish to store, and $\mathcal{S}(\boldsymbol{h})$ contains the corresponding 1-hot representation of each target. Thus, the weights are learned for *all* target memories. Gradient descent on $\boldsymbol{\xi}$ then leads to

$$\tau_\xi \frac{d\boldsymbol{\xi}}{dt} = \boldsymbol{v}\ \mathcal{S}(\boldsymbol{h}), \tag{3.6}$$

where τ_ξ is a time constant that determines the rate at which $\boldsymbol{\xi}$ is learned.

A penalty term, $\|\boldsymbol{\xi}\|_F^2$ – based on the Frobenius norm of the weight matrix – is added to equation (2.15) to discourage large connection weights. Gradient descent on that term with respect to $\boldsymbol{\xi}$ introduces a weight-decay term to equation (3.6),

$$\tau_\xi \frac{d\boldsymbol{\xi}}{dt} = \boldsymbol{v}\ e^{\boldsymbol{f}}\ -\ \boldsymbol{\xi}, \tag{3.7}$$

where we have replaced $\mathcal{S}(\boldsymbol{h})$ with $e^{\boldsymbol{f}}$ to implement the learning in the locLSE network.

Hence, if the feature and hidden nodes are clamped to their ideal values for a given set of target memories, the locLSE network is able to *learn* the connection weights in order to

store these targets using the equations (3.1 - 3.4) as well as (3.7), instead of having them prescribed *a priori*. To ensure this learning occurs correctly we require that $\tau_s \ll \tau_\xi$. This makes sure that the softmax sub-network reaches equilibrium almost immediately, which then pushes $\boldsymbol{\xi}$ to the correct weights. Recall that the other set of connection weights are symmetric to these values and so we can use the same learning rule here, however, the $\boldsymbol{f}$ nodes are not local to those connections, and so $\boldsymbol{\xi}^T$ essentially constitutes weight copying. Once these weights are learned, we clamp the weights to these learned values and unclamp the feature and hidden nodes. This is done by initializing $\boldsymbol{\xi}$ as these learned weights and clamping equation (3.7) to zero so they don't change. Then, all other nodes are initialized near zero and equations (3.1 - 3.4) are run to test how the network behaves when provided an input pattern, $\boldsymbol{I}$.

3.3 Network Behaviour

Here, we test the locLSE network and show that it can behave as a CAM. In other words, it can successfully store target patterns and retrieve them from memory when prompted with noisy or corrupted versions. It is important to note that, in all experiments in this work, the `solve_ivp` function from the SciPy python library is used to solve the systems of differential equations; more specifically, the Runge-Kutta integration method is used.

Consider a memory dataset, $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$ of $M = 20$ binary targets, each with $D = 15$ bits. We can first learn the connection weights, $\boldsymbol{\xi}$ from $\boldsymbol{f}$ to $\boldsymbol{v}$, using the process outlined in section 3.2. That is, $\boldsymbol{v}$ and $\boldsymbol{h}$ are set to their ideal values and clamped there ($d\boldsymbol{v}/dt = 0, d\boldsymbol{h}/dt = 0$), while $\boldsymbol{\xi}$ is initialized randomly and changes according to equation (3.7). We learn the weights for a total of 1 second, with time constants set to $\tau_\xi = 1\ ms$, and $\tau_s = 0.1\ ms$. After the weights are learned, we move into the test phase where $\boldsymbol{\xi}$ is set to these learned weights and clamped there ($d\boldsymbol{\xi}/dt = 0$). Then, all other vectors are initialized randomly near zero and change according to equations (3.1 - 3.4).

Suppose we construct an input pattern $\boldsymbol{I} \in \{-1, 1\}^{D \times 1}$ and compute $\texttt{ham}(\boldsymbol{I})$, the hamming distance between $\boldsymbol{I}$ and each of the M targets, and get

$$\texttt{ham}(I) = [4,\ 6,\ 8,\ 6,\ 8,\ 5,\ 7,\ 9,\ 7,\ 5,\ 8,\ 9,\ 5,\ 7,\ \mathbf{3},\ 4,\ 4,\ 6,\ 8,\ 9].$$

Clearly, the target pattern closest to (or "most similar to") $\boldsymbol{I}$ differs from it by **3** bits and corresponds to target pattern $m = 14$. Figure 3.4 shows the dynamics of the locLSE network when it is run for 2 seconds with the input turned on for the first second, and time constant values of $\tau_v = \tau_h = 10\ ms, \tau_s = 1\ ms$. The feature nodes promptly correct the 3 bits and converge to the 14^{th} pattern, while the hidden nodes converge to the "pre-one-hot" representation of this target, where the index of the closest target is much larger than all other values. This representation is known as the *logit*, which consists of unnormalized probability values, usually referring to the inputs to the softmax function. That is, the softmax of the converged (or"final") hidden state, $\boldsymbol{h}_{\text{final}}$, yields the one-hot representation of target 14:

$$\mathcal{S}(\boldsymbol{h}_{\text{final}}) = [0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0.0025,\ 0,\ 0,\ 0,\ 0,\ \mathbf{0.9974},\ 0,\ 0,\ 0,\ 0,\ 0].$$

This example run is meant to simply illustrate how the locLSE network can behave as a CAM. Further analysis of this behaviour is presented later in chapter 5.

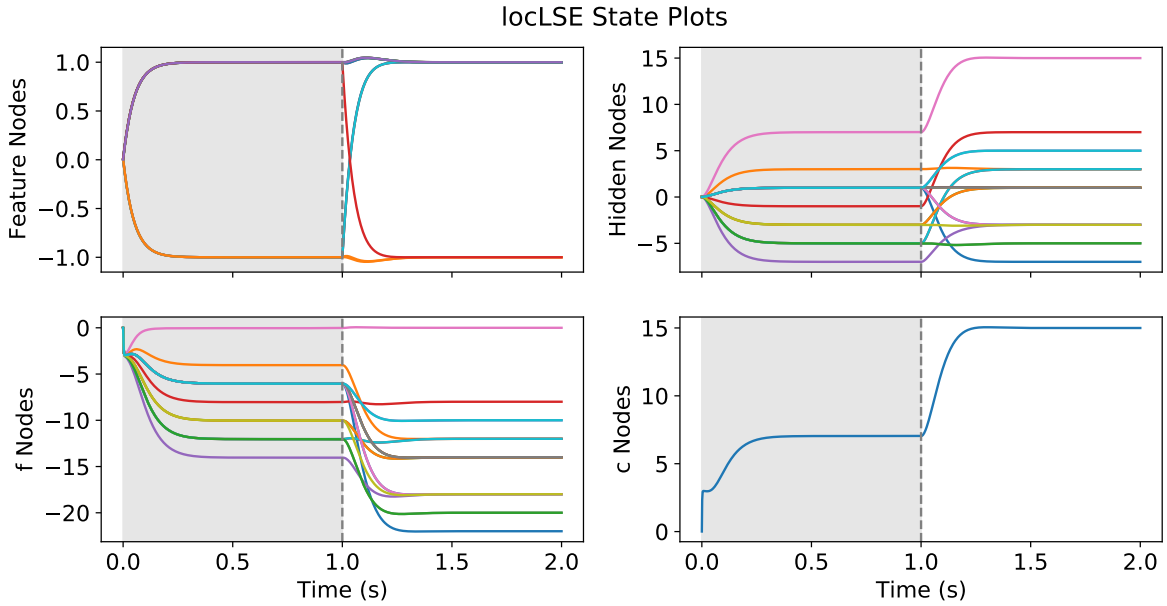


Figure 3.4: The dynamics of all groups of nodes in the locLSE network that stores $M = 20$ targets with $D = 15$ bits each. The plots show the behaviour of the network for a single run, using a single perturbed input, $\boldsymbol{I}$. The shaded region marks where $\boldsymbol{I}$ is turned on. Each line represents a different node in a given group.

Chapter 4

Local LSE Network with Distributed Hidden Representation

One aspect of the locLSE network that remains biologically suspicious is the reliance on a one-hot hidden representation. If each target in our memory dataset relies on the activity of a single hidden node, this suggests that for a given memory or idea, there is a single neuron in the brain that activates to represent it. There exist many theories of such a "Grandmother cell", i.e., a single neuron that represents a specific concept [14]. However, it is largely accepted that instead, groups (or populations) of neurons collectively encode external or sensory input [1, 12, 32]. A natural next step is to apply this population/neural coding to the hidden layer of the locLSE network. To this end, the NEF presented in section 2.1.1 is applied to the hidden layer. Here, we present the *Distributed locLSE network*, which involves a distributed hidden representation via the NEF in addition to local computations. It is important to note that, in this network, *only* the hidden nodes will have a distributed neural representation. A visual of the Distributed locLSE network can be seen in Figure 4.1 (c).

First, let us consider $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$ to be the memory dataset consisting of M target patterns, each with D bits. The ideal activities of the feature neurons are these target patterns, which we can hold in each column of $\boldsymbol{v} = \boldsymbol{X}$. The ideal hidden activities for each target are thus held in the columns of $\boldsymbol{h} = \boldsymbol{X}^T\boldsymbol{v}$. The hidden representation

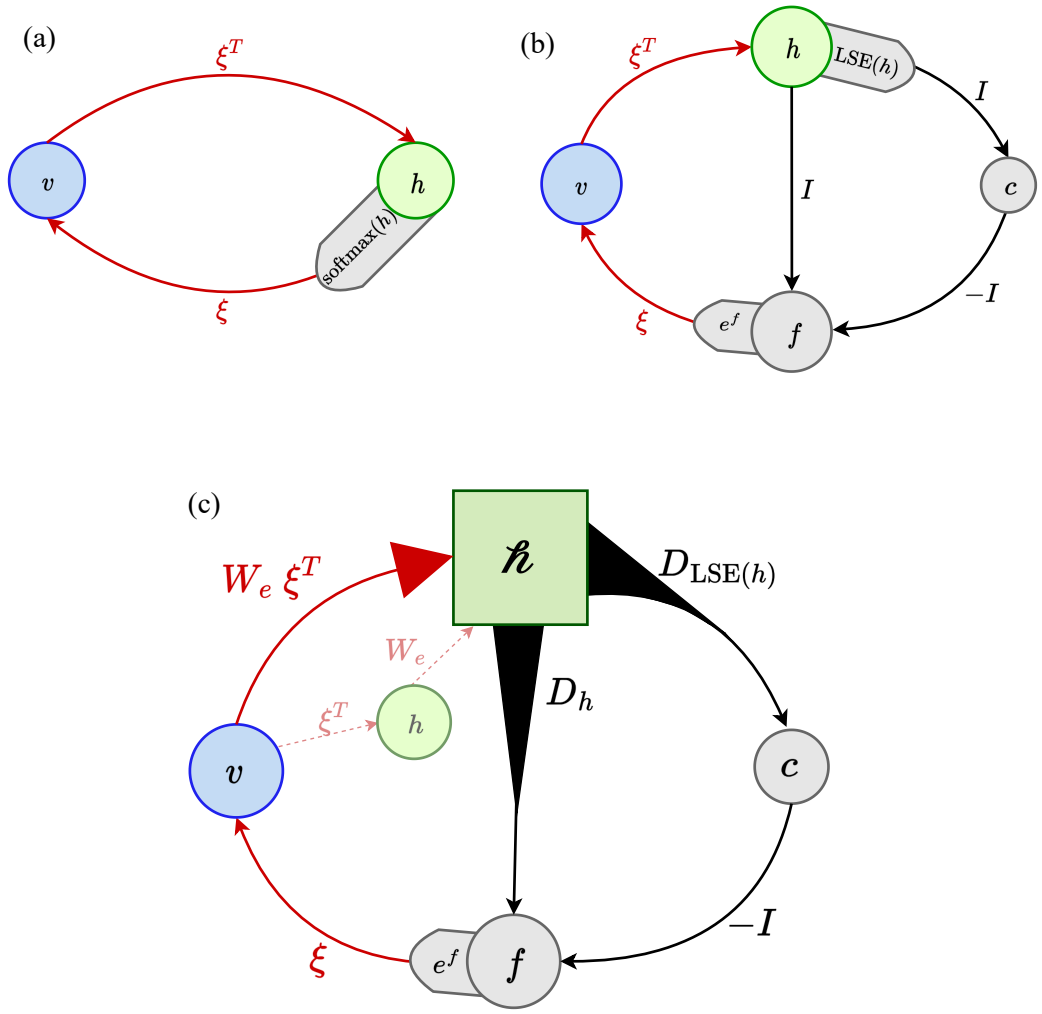


Figure 4.1: A diagram of (a) the LSE network from [20], (b) the locLSE network, and (c) the Distributed locLSE network. Each circle represents a node or group of nodes, while each square represents an ensemble of neurons. Specifically, the blue $\boldsymbol{v}$ circle represents a group of N_v nodes, while the $\boldsymbol{\hbar}$ square represents an ensemble of $N_{\hbar}$ neurons. The red connection arrow with the larger arrow head represents a connection that encodes/projects values into a higher dimensional ensemble of neuron. The black connection arrows with wide bases represent connections that decode out certain values from the higher dimensional neural representations.

for each target, $\boldsymbol{h}$, can be thought of as the logit of each target, i.e. a "pre-1-hot" or "winner-takes-all" representation, where the index corresponding to the target is larger (more active) than all other entries. Hence, when $\boldsymbol{h}$ is fed through the softmax function, we get the 1-hot representation of each target, i.e. all values are normalized with respect to each other. This is a convenient way to represent data, however it is not biologically accurate. Having a hidden layer consisting of *many* neurons that collectively represent each target is more biologically realistic than having a single node per target, which the current 1-hot representation suggests. Hence, we wish to represent $\boldsymbol{h}$ in a higher dimensional ensemble, $\boldsymbol{\hbar}$. This ensemble will give a more combinatorial representation of each target amongst $N_{\hbar}$ neurons, and will not rely on a single neuron per target pattern.

More specifically, we wish to encode the hidden representations, $\boldsymbol{h}$, in the columns of $\boldsymbol{\hbar} \in \mathbb{R}^{N_{\hbar} \times M}$. From $\boldsymbol{\hbar}$, we want be able to decode $\boldsymbol{h}$ and LSE($\boldsymbol{h}$) to pass to $\boldsymbol{f}$ and c respectively, as seen in Figure 4.1. Moreover, we want to solve for weights, $\boldsymbol{D}_{\boldsymbol{h}}$ and $\boldsymbol{D}_{\text{LSE}(\boldsymbol{h})}$ that can linearly decode these values.

Following the NEF, we first create a bunch of target samples, $\boldsymbol{X}_s \in \{-1, 1\}^{D \times N_s}$, where N_s is the number of samples. These samples consist of copies of the target memories, as well as noisy versions of them to ensure the network can handle perturbations. Next, we can clamp the feature nodes to these samples, $\boldsymbol{v}_s = \boldsymbol{X}_s$. Similarly, we clamp the hidden nodes for this set of samples to $\boldsymbol{h}_s = \boldsymbol{X}^T \boldsymbol{v}_s$ which is then encoded into $\boldsymbol{\hbar}_s$. Note that we could learn the decoders for each sample one at a time by clamping the feature and hidden nodes to their ideal values for each target memory and cycling through them. However, here we implement things batch-wise using matrices, so that the feature and hidden nodes are clamped to their ideal values for the entire memory dataset. Thus, according to the NEF from section 2.1.1, the distributed hidden representation for all targets is computed as follows,

$$\boldsymbol{\hbar}_s = G\left[\boldsymbol{W}_e \boldsymbol{h}_s + \boldsymbol{b}_e\right], \tag{4.1}$$

where $\boldsymbol{W}_e, \boldsymbol{b}_e \in \mathbb{R}^{N_{\hbar} \times M}$ are the encoding weights and biases respectively. In our studies, we use real-valued encoding weights and biases, however one can use binary encoding weights as well.

Next, we solve for the decoding weights for the sampled version of this problem de-

scribed above. Specifically, we wish to find decoding weights that approximately solve $\boldsymbol{\hbar}_s^T \boldsymbol{D}_{\boldsymbol{h}}^T = \boldsymbol{h}_s^T$ and $\boldsymbol{\hbar}_s^T \boldsymbol{D}_{\mathrm{LSE}(\boldsymbol{h})}^T = \mathrm{LSE}(\boldsymbol{h}_s)^T$. These equations are written in the form $AX = B$, where we want to solve for X, so that we can easily find the least square solution. Using the least square solution gives $\boldsymbol{D}_{\boldsymbol{h}} = (\boldsymbol{\hbar}_s \boldsymbol{\hbar}_s^T)^{-1} \boldsymbol{\hbar}_s \boldsymbol{h}_s^T$ and $\boldsymbol{D}_{\mathrm{LSE}(\boldsymbol{h})} = (\boldsymbol{\hbar}_s \boldsymbol{\hbar}_s^T)^{-1} \boldsymbol{\hbar}_s \mathrm{LSE}(\boldsymbol{h}_s)^T$.

Thus, we can write equations (3.1 - 3.4) using a distributed hidden representation as follows,

$$\tau_v \frac{d\boldsymbol{v}}{dt} = (1-\beta)\boldsymbol{\xi} e^{\boldsymbol{f}} - \boldsymbol{v} + \beta \boldsymbol{I} \tag{4.2}$$

$$\tau_h \frac{d\boldsymbol{\hbar}}{dt} = G(\boldsymbol{W}_e(\boldsymbol{\xi}^T \boldsymbol{v}) + \boldsymbol{b}_e) - \boldsymbol{\hbar} \tag{4.3}$$

$$\tau_s \frac{dc}{dt} = \boldsymbol{D}_{\mathrm{LSE}(\boldsymbol{h})} \boldsymbol{\hbar} - c \tag{4.4}$$

$$\tau_s \frac{d\boldsymbol{f}}{dt} = \boldsymbol{D}_{\boldsymbol{h}} \boldsymbol{\hbar} - c - f. \tag{4.5}$$

where G can be any non-linear function; here we use the logistic function, $\sigma(x) = (1 + e^{-x})^{-1}$. A diagram of all this Distributed locLSE network, along with the previous two networks, can be seen in Figure 4.1.

4.1 Learning Connection Weights

The connection weight matrix $\boldsymbol{\xi}$ can also be learned here using the same procedure as presented in section 3.2. That is, after we know our decoders, we clamp $\boldsymbol{v}$ and $\boldsymbol{\hbar}$ to their ideal values for a set of target memories, and adjust the weights to minimize the energy. The difference here is that the hidden representation is not clamped to the logit representation seen in the locLSE network, instead it is clamped to the distributed hidden representation. For example, take $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$ to be a memory dataset of M targets with D bits each. Similar to the learning in the locLSE, we can clamp the feature nodes to the target memories by setting $\boldsymbol{v} = \boldsymbol{X}$ and $d\boldsymbol{v}/dt = 0$. However, the hidden neurons are clamped to the distributed representation of these targets by setting $\boldsymbol{\hbar} = \sigma(\boldsymbol{W}_e(\boldsymbol{X}^T \boldsymbol{v}) + \boldsymbol{b}_e)$ and $d\boldsymbol{\hbar}/dt = 0$. To implement the learning, we use equation (3.7), i.e. the same rule derived for the locLSE network.

4.2 Network Behaviour

Here, we illustrate how the Distributed locLSE network can *also* act as a CAM by using the same example used on the locLSE network in section 3.3. Namely, we use the same memory dataset, $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$ of $M = 20$ targets, each with $D = 15$ bits, as well as the same perturbed input $\boldsymbol{I}$. The hidden ensemble, $\boldsymbol{\hbar}$, consists of $N_{\hbar} = 2000$ neurons. Again, we first learn the weights from $\boldsymbol{f}$ to $\boldsymbol{v}$, $\boldsymbol{\xi}$, using the process described in section 4.1. Here, we learn the weights over a total time of $1s$, with time constant values of $\tau_{\xi} = 1\ ms$, and $\tau_s = 0.01\ ms$. Once the weights are learned, we move into the test phase, initializing $\boldsymbol{\xi}$ as these learned weights and clamping them by setting $d\boldsymbol{\xi}/dt = 0$. Then, all other nodes/neurons are initialized randomly near zero and change according to equations (4.2 - 4.5).

Since the same input and memory dataset are used here, we expect this network to converge to the 14^{th} target pattern like the locLSE did; this is the target in memory that is most similar to $\boldsymbol{I}$. The network is ran for 2 seconds, with the input turned on for the first second, and time constant values of $\tau_v = \tau_h = 1\ ms$ and $\tau_s = 0.05\ ms$. The dynamics of the network during this process can be seen in Figure 4.2. The feature nodes converge to pattern 14, fixing the 3 "incorrect" bits just like the feature nodes of the locLSE network did. The hidden representation converges to a more combinatorial representation of this target across 2000 neurons, as opposed to the pre-one-hot representation of this target as in the locLSE network. This example illustrates that the Distributed locLSE network can act as a CAM, with dynamics similar to the locLSE network. A deeper, more statistical comparison of the three networks (the LSE, locLSE, and Distributed locLSE) is done later in chapter 5.

4.2.1 Dropout Encoded Hidden Neurons

The Distributed locLSE network does not rely on a single neuron in the hidden layer per target pattern, as is the case in the LSE and locLSE networks. This means that we can "drop" or "kill" some neurons in the hidden layer, and the network should still be able to operate as a CAM. We can illustrate this by turning off a certain amount of the $\boldsymbol{\hbar}$ neurons,

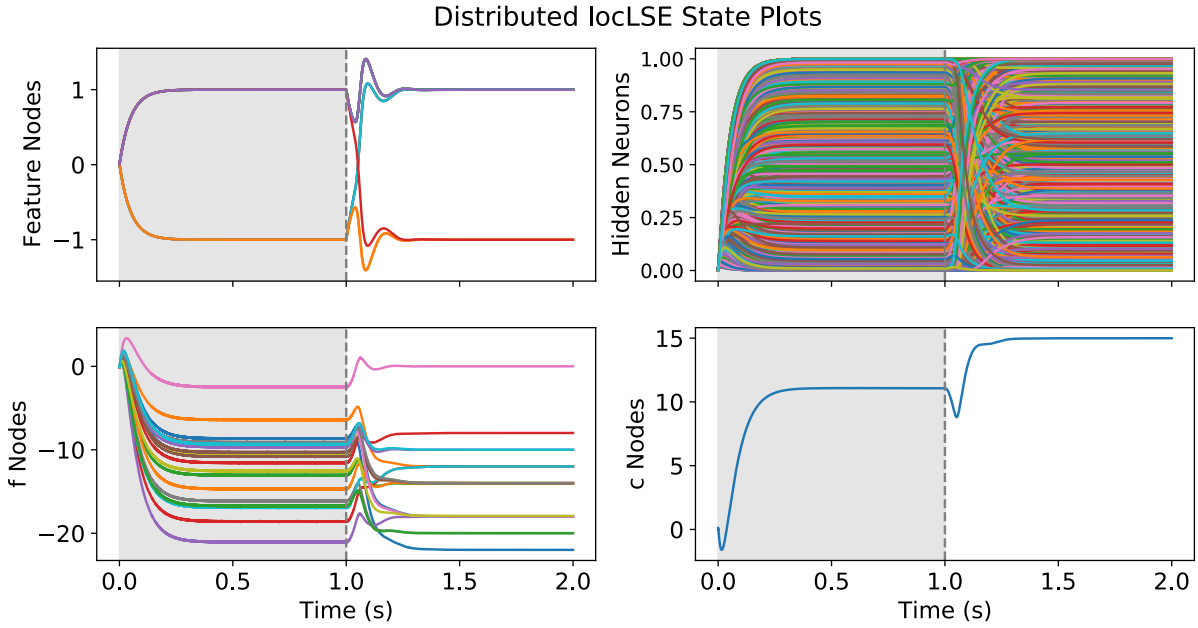


Figure 4.2: The dynamics of all groups nodes/neurons in the Distributed locLSE network that stores $M = 20$ targets, each with $D = 15$ bits. The plots show the behaviour of the network for a single run, using a single perturbed input, $\boldsymbol{I}$. Each line represents the activity of a different node/neuron in a given group. The shaded region marks where $\boldsymbol{I}$ is turned on.

and showing that it still acts as expected. That is, we can show that the Distributed locLSE with $\boldsymbol{\hbar}$ neurons dropped still converges to the target memory closest to the given input. Specifically, to "drop" some $\boldsymbol{\hbar}$ neurons, we make them inactive by initializing them at 0, and clamping them there by setting $d\boldsymbol{\hbar}/dt = 0$ for these neurons. It's important to note that the encoded hidden representation of target memories involve neurons that are already inactive, i.e., some neurons are already zero (or near zero) in their representations. So, dropping these inactive neurons would be redundant in this experiment. Hence, when we are choosing the $\boldsymbol{\hbar}$ neurons to drop, we make sure that we drop neurons that are *active* (non-zero).

Consider the same memory dataset, $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$, used in the previous examples, which consists of $M = 20$ target memories, each with $D = 15$ bits. As done before, we

first learn the connection weight matrix, $\boldsymbol{\xi}$, for this memory dataset, and then construct a separate input pattern $\boldsymbol{I}$. Suppose we check which target memory is closest to this input and get that the m^{th} target memory is the closest at 4 bits away. Next, we dropout 10% of the 2000 neurons in $\boldsymbol{\hbar}$ that are active when representing the m^{th} target memory. The Distributed locLSE network is ran for 2 seconds total, with $\boldsymbol{I}$ turned on for the first second, and time constant values of $\tau_v = \tau_h = 1\ ms$, and $\tau_s = 0.05\ ms$. To verify that this network behaves as it should, we perform this same run on the LSE network, which uses time constant values of $\tau_v = \tau_h = 1\ ms$. Figure 4.3 shows the behaviour of (a) the LSE network and (b) the Distributed locLSE network during this run. Both the LSE and the Distributed locLSE network with dropped hidden neurons behave similarly here; the feature nodes of both networks converge to the closest target.

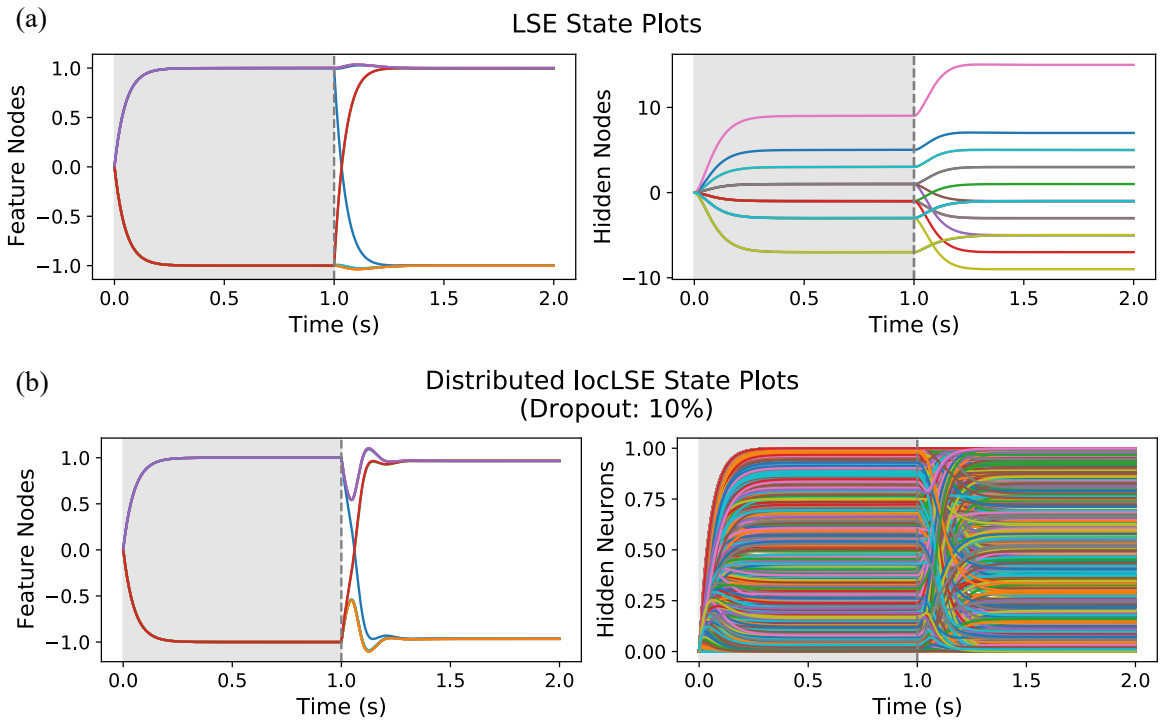


Figure 4.3: The dynamics of (a) the LSE network and (b) the Distributed locLSE when storing the same memory dataset and initialized with the same input pattern, $\boldsymbol{I}$. The evaluation of the feature nodes over time are shown for both networks on the left, as well as the hidden nodes/neurons on the right. Here, 10% of the distributed hidden neurons are dropped out in the Distributed locLSE. Each line represents a different node/neuron in a given group. The shaded region marks where $\boldsymbol{I}$ is turned on.

Chapter 5

Content-Addressable Memory Experiment

We have seen in sections 3.3 and 4.2 that the locLSE and Distributed locLSE networks behave as CAMs; both networks can successfully correct a perturbed input and converge to the closest target from memory. However, here we directly compare how the locLSE and Distributed locLSE networks compare to the original LSE network in terms of this behaviour. The goal is not to illustrate that the locLSE and Distributed locLSE networks perform better than the original LSE network, but rather to demonstrate that these two networks perform similarly to the original, while using more biologically-practical methods. To compare the three networks of interest we use the *Content Addressable Memory (CAM) Experiment*, which is described here. Again, note that the `solve_ivp` function (using the Runge-Kutta integration method) from the SciPy python library is used to solve the systems of differential equations in these experiments.

Given M, D, and N_h, the first step in this experiment is to create a memory dataset of M targets with D bits each, i.e., $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$. Next, create the three networks of interest: the original LSE network, the locLSE network, and the Distributed locLSE network which has a hidden layer consisting of N_h neurons. Then, the weights, $\boldsymbol{\xi} \in \{-1, 1\}^{D \times M}$, are learned in the locLSE network by applying the learning rules derived in section 3.2. The Distributed locLSE network also learns its connection weights by

Algorithm 1 Content Addressable Memory (CAM) Experiment

Input: M, D, and $N_{\not\ell}$

$iter = 1$

$c_{\text{LSE}}, c_{\text{locLSE}}, c_{\text{DlocLSE}} = 0$

while $iter < 100$ **do**

Create memory dataset: $\boldsymbol{X} \in \{-1, 1\}^{D \times M}$

Create networks: LSE, locLSE, and Distributed locLSE

Learn weights for the locLSE and Distributed locLSE networks

Create input pattern: $\boldsymbol{I} \in \{-1, 1\}^{D \times 1}$

Compute the target(s) in memory closest to $\boldsymbol{I}$: $closest\_targs$

Run all networks for 2 seconds with $\boldsymbol{I}$ on only for the first half

Compute which targets each of the networks converge to: $m_{\text{LSE}}, m_{\text{locLSE}}$, and m_{DlocLSE}

if m_{LSE} is in $closest\_targs$ **then**

$c_{\text{LSE}} \leftarrow c_{\text{LSE}} + 1$

end if

if m_{locLSE} is in $closest\_targs$ **then**

$c_{\text{locLSE}} \leftarrow c_{\text{locLSE}} + 1$

end if

if m_{DlocLSE} is in $closest\_targs$ **then**

$c_{\text{DlocLSE}} \leftarrow c_{\text{DlocLSE}} + 1$

end if

$iter \leftarrow iter + 1$

end while

Return $c_{\text{LSE}}/iters$, $c_{\text{locLSE}}/iters$, $c_{\text{DlocLSE}}/iters$

Output: The percentage of time that the LSE, locLSE and Distributed locLSE networks converge to the target in memory that is closest to the perturbed input.

applying the rules from section 4.1. Note that the connection weight matrices learned by the locLSE and Distributed locLSE networks are learned separately and are potentially different matrices. The weights for the LSE network are set directly to the target memories, not learned. Next, a separate input pattern, $\boldsymbol{I} \in \{-1, 1\}^{D\times 1}$, is constructed by randomly selecting one of the targets in memory and perturbing it by flipping a certain amount of bits. Then, we compute *closest_targ* which collects the target(s) from memory that is closest (i.e., has the smallest hamming distance) to $\boldsymbol{I}$. Note that *closest_targ* isn't always the randomly selected target that we perturbed; after these perturbations, $\boldsymbol{I}$ may be just as close or closer to another target in memory. Next, we run all three networks for two seconds, with $\boldsymbol{I}$ turned on for the first second, and using the same time constants as in the previous examples. After the two seconds, we check which target from memory each of the networks converged to and denote the index of the closest target using, $m_{\text{LSE}}, m_{\text{locLSE}}, m_{\text{DlocLSE}}$. Then, we check if the networks converged to the *closest_targ* (i.e., check if $m_{\text{LSE}}, m_{\text{locLSE}}$, and m_{DlocLSE} are in *closest_targ*). If a given network does converge to the *closest_targ*, we mark this as a success and keep a count of each successful iteration for each network, $c_{\text{LSE}}, c_{\text{locLSE}}, c_{\text{DlocLSE}}$. This entire process is carried out 100 times. Once completed, the experiment returns the percentage of time each network successfully converged to the *closest_targ*. The steps involved in the experiment are also outlined in Algorithm 1.

This experiment provides a more direct comparison of the behaviour of all three networks since we are evaluating them on the same memory dataset and input in each iteration. When we ran the experiment, we hoped to see that all three networks converge to the nearest target at similar rates. Such results would verify that the biological methods involved in the locLSE and Distributed locLSE networks do not impede on their ability to act as a CAM. To make this comparison even more extensive, we ran the CAM experiment for a variety of memory dataset sizes. The results of these experiments can be seen in Table 5.1.

$(M, D, N_{\hbar})$ $\%_{\text{avg. pert}}$	LSE	locLSE	Dist. locLSE
$(4, 10, 1000)$ 35%	100%	100%	92%
$(10, 10, 1000)$ 28%	96%	96%	90%
$(15, 12, 1000)$ 27%	98%	98%	94%
$(18, 12, 2000)$ 28%	100%	100%	98%
$(20, 12, 2000)$ 29%	95%	95%	90%
$(20, 15, 2000)$ 27%	98%	98%	94%
$(25, 18, 3000)$ 26%	99%	99%	93%
$(30, 24, 8000)$ 28%	99%	99%	94%
$(32, 22, 8000)$ 30%	99%	99%	94%
$(45, 35, 8000)$ 30%	99%	99%	95%

Table 5.1: The results from the CAM experiment for different memory dataset sizes. The results listed under each network represent the rate at which each network converged to the target closest to a given input, $\boldsymbol{I}$, for the given memory dataset size (M, D). For each experiment run, we also report the number of distributed hidden neurons used ($N_{\hbar}$), and the average level of perturbation between $\boldsymbol{I}$ and the closest target memory ($\%_{\text{avg. pert}}$).

An example of a single iteration from a CAM experiment can be seen in Figure 5.1. In the experiment that this iteration is taken from, a memory dataset of $M = 25$ patterns, each with $D = 18$ bits is used, and $N_{\hbar} = 3000$ neurons are used for the hidden ensemble in the Distributed locLSE network. In this specific iteration, the input, $\boldsymbol{I}$, differs from the closest target pattern by 4 bits. Each network is able to correct these bits and converge fully to the closest target pattern in this case. This iteration shows the three networks *successfully* behaving as a CAM. However, there are failed iterations in which the locLSE and/or the Distributed locLSE networks converge to a target that is not the closest to the

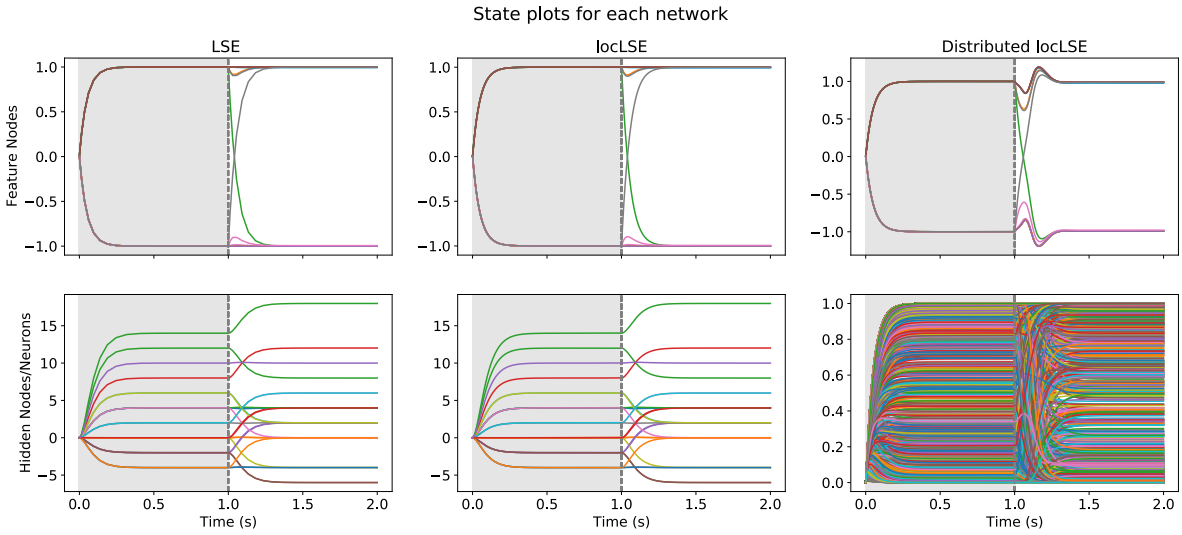


Figure 5.1: The dynamics of the LSE, locLSE, and the Distributed locLSE networks during a single iteration of a CAM experiment. In this example, the networks were storing the same $M = 25$ targets, each with $D = 18$ bits. $N_{\not h} = 3000$ neurons are used for the hidden ensemble in the Distributed locLSE network. Each network is given the same input pattern $\boldsymbol{I}$. The top row shows the feature dynamics for each network and the bottom row shows the hidden dynamics for each network. Each coloured line represents a different node/neuron in a given group. The shaded region marks where the input $\boldsymbol{I}$ is turned on.

input. In these cases, the networks still converge to one of the target patterns, usually the second closest. There also exist iterations in which there are multiple targets closest to the input. Sometimes in these cases the Distributed locLSE network would converge to a different closest target than the LSE and locLSE networks.

Chapter 6

Nengo Implementation

Nengo is a software tool used to construct and simulate models based on the NEF [2]. It is based on the Python library, Nengo, and a numpy-based simulator. The Nengo library includes the five classes that represent nodes, populations of neurons (ensembles), connections, probes, and networks.

Naturally, the next step to make this network even more biologically realistic is to have a distributed representation of all layers rather than just the hidden layer. Implementing the LSE or the locLSE network using Nengo causes *all* groups of nodes to be encoded into a distributed representation across populations of neurons.

The first step in implementing these networks in Nengo is to create the network objects. Within each network, ensembles are created to represent the groups of nodes. For example, when creating the locLSE network in Nengo, we must create four ensembles of neurons to represent the $\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{f}$, and c nodes. An ensemble object is a group of neurons that collectively represent a vector. The user specifies how many neurons are in this group, and the dimensions of the vector we want to represent. For example, $\boldsymbol{v}$ is set to be an ensemble object with a certain number of neurons that represents a D dimensional vector. The user can also specify the type of neurons used in each ensemble, i.e., LIF neurons.

The node object is used to provide non-neural inputs to other Nengo objects. In the case of both the LSE and the locLSE model, a node is used to provide the input, $\boldsymbol{I}$, to

the feature ensemble. A node can also be used to inhibit neurons. For example, we used a node to inhibit the the $\boldsymbol{f}$ ensemble when the input is on, mimicking the purpose of the β factor in equation (3.1).

Using the connection object, we can make the necessary connections between the ensembles to create the locLSE network. Within the connection object, the user can specify a function to be computed across said connection. For example, when creating the locLSE network in Nengo, the $\boldsymbol{h}$ ensemble is connected to the c ensemble and the function computed across this connection is LSE($\boldsymbol{h}$). The rest of the connections can be seen in Figure 6.1.

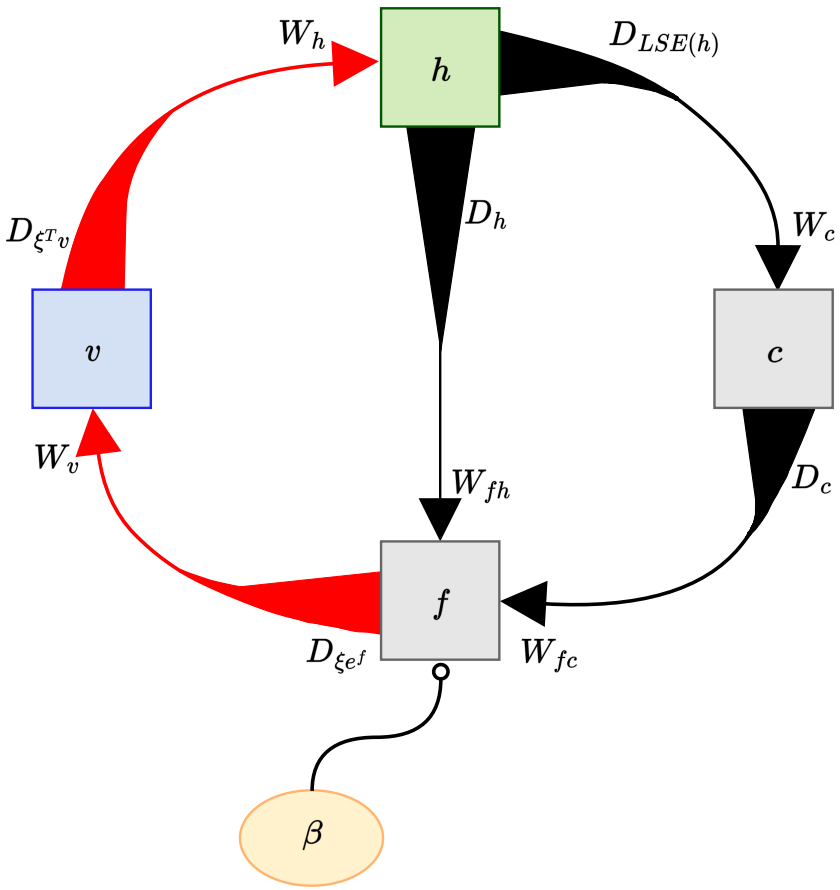


Figure 6.1: A diagram of the locLSE network in Nengo, showing how the NEF is applied to all groups of nodes. Each square is an ensemble of neurons that represents the labeled data, i.e., the $\boldsymbol{v}$ square is an ensemble of neurons which collectively represent the $\boldsymbol{v}$ nodes in a higher dimensional space. The arrows with larger arrow heads represent encoding connections, while the arrows with larger bases represent decoding connections. The oval labelled β represents a node that inhibits the neurons in the f population when the input is turned on.

Finally, we create probe objects for each ensemble. A probe object essentially allows the user to collect data from an ensemble during the simulation. This allows us to visualize the behaviour of these ensembles and their decoded outputs.

By implementing the locLSE network in Nengo, we are essentially applying the NEF to the entire network in the sense that ALL groups of nodes ($\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{f}$, and c) are represented by a higher dimensional ensemble of neurons. Nengo provides the option to learn a communication channel (connection weight) based on different learning rules. As previously mentioned, Nengo gives the choice of what neuron model to use for the nonlinear encoding, i.e., the types of neurons in the ensembles. Thus, the networks created in Nengo are not limited to or dependent on a single neuron model. In our analysis we focus on the LIF model (described in section 2.1.1). Within this model we experiment with both *LIF-rate* and *LIF* neurons. The biggest difference between these two types of neurons is the amount of noise they involve. LIF-rate neurons encode values based on the preconceived average firing rate of the neurons, i.e., equation (2.2). The LIF neurons (LIF) still encode values using the firing-rate, however they manifest the firing rate of the neurons by simulating actual spikes and then taking the average; this induces more noise into the system.

6.1 Network Behaviour

Once we have constructed the LSE and locLSE networks in Nengo, we analyse their behaviours by applying the CAM experiment. Note that in each iteration we only evaluate the performance of the LSE and locLSE networks in Nengo, since these are the networks of interest here.

It was found that the locLSE network can successfully act as a CAM for small memory datasets. Although, even in these cases, very large numbers of neurons are required. For example, consider a small memory dataset of $M = 4$ targets, each with $D = 10$ bits. We encode the $\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{f}$, and c nodes in ensembles of $5000, 6000, 6000$, and 3000 LIF-rate neurons respectively. When the CAM experiment is ran, the locLSE network in Nengo is found to converge to the target closest to the input pattern **90**% of the time. An example of a single iteration from this experiment can be seen in Figure 6.2. In this specific iteration, the target in memory that is closest to $\boldsymbol{I}$ differs from it by 3 bits. We can see that the network is able to successfully act as a CAM and fully converge to this closest target in this case. Interestingly, the original LSE network in Nengo only converges to the nearest

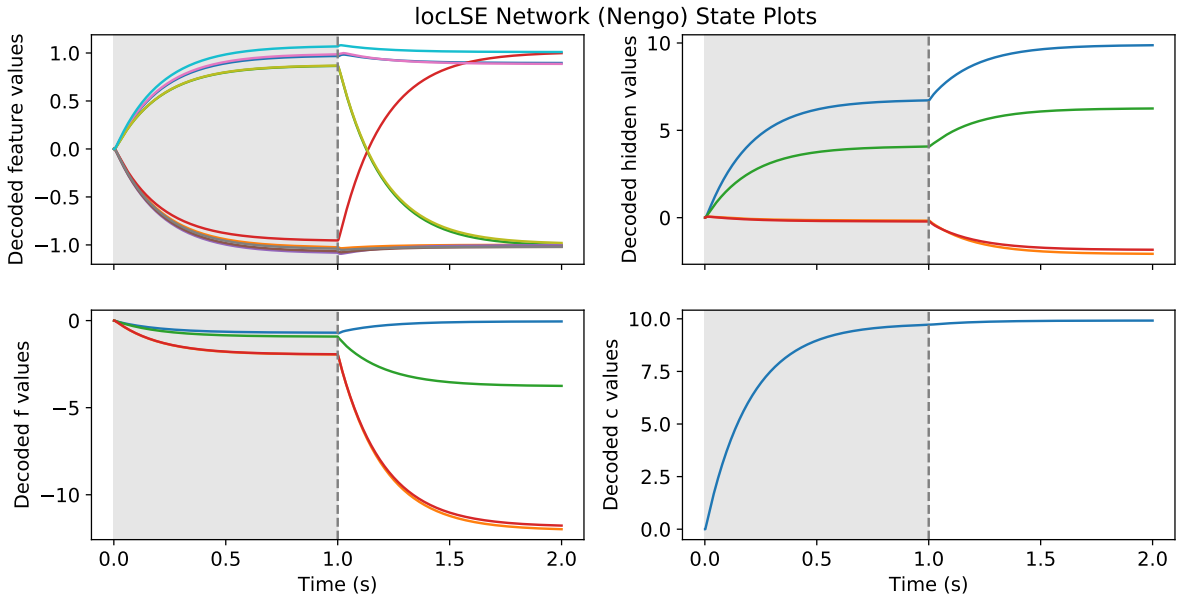


Figure 6.2: The dynamics of the locLSE network in Nengo when ran for 2 seconds with a perturbed input $\boldsymbol{I}$ turned on for the first second. The lines represent the decoded activities from each corresponding ensemble of LIF-rate neurons.

target **60**% of the time in the same CAM experiment. The original LSE network seems to have more trouble decoding the softmax values when implemented in Nengo.

We also tested how the locLSE and original LSE networks behave when using spiking neurons (LIF rather than LIF-rate neurons). If we run the CAM experiment using the same sized dataset ($M = 4, D = 10$) and the same number of LIF neurons in each ensemble, we find that the locLSE network converges to the target closest to $\boldsymbol{I}$ **72**% of the time, while the original LSE network does so **54**% of the time. Hence, in Nengo, the locLSE network acts as a CAM more often than the original LSE for small memory datasets, when using a sufficient amount of neurons. This holds for both LIF and LIF-rate neurons.

However, the locLSE network implemented in Nengo does not perform as efficiently as a CAM when we introduce larger memory datasets. For example, consider a slightly larger memory dataset, this time consisting of $M = 10$ targets, each with $D = 10$ bits. When we ran the CAM experiment using the *same number* of LIF-rate neurons that were

used for the smaller dataset, we found that the locLSE network converged to the target closest to $\boldsymbol{I}$ just **64**% of the time. This is still better than the performance of the original LSE network for this CAM experiment, which only converged to the nearest target **52**% of the time. However, if we run the CAM experiment on the locLSE and original LSE networks for this larger dataset, but increase the number of neurons in each ensemble by 2000 neurons, we found that this increased the rate at which the locLSE and original LSE networks converged to the closest target to **76**% and **68**% respectively. This suggests that, in Nengo, the locLSE network can still behave as a CAM for larger memory datasets, given a sufficiently large amount of neurons.

Chapter 7

Discussion

The objective of this work was to extend on the biological plausibility of Krotov and Hopfield's LSE Hopfield network (the LSE network). Figure 7.1 outlines the progression of the LSE network towards a more biological implementation. We then wanted to verify that making this network more biological doesn't interfere with its ability to act as a CAM and model associative memory.

The first step in making the LSE network more biological was to implement the softmax function using neurally local computations, resulting in our locLSE network. Figure 4.1 shows a diagram of all 3 networks studied in this work. It outlines that the locLSE network is created by replacing the softmax function in the LSE network with a sub-network that computes the softmax using only local computations. In addition to this, we also *learned* the network's connection weight matrix, $\boldsymbol{\xi}$. In section 3.3 we illustrate a test run of this network, first creating a memory dataset and then learning the connection weights for all of these targets. A separate input, $\boldsymbol{I}$, is constructed and used as input for 1 second, which is then turned off and the network is then ran to equilibrium for another second. It is clear from Figure 3.4 that, once $\boldsymbol{I}$ is turned off, the feature nodes are pushed to converge to the equilibrium state corresponding to the closest target pattern. This is because when the input is turned off, the hidden neurons take control and push the feature state to the target with the largest softmax output, i.e., the target most similar to the input pattern.

The results from the CAM experiments in Table 5.1 show that the locLSE network

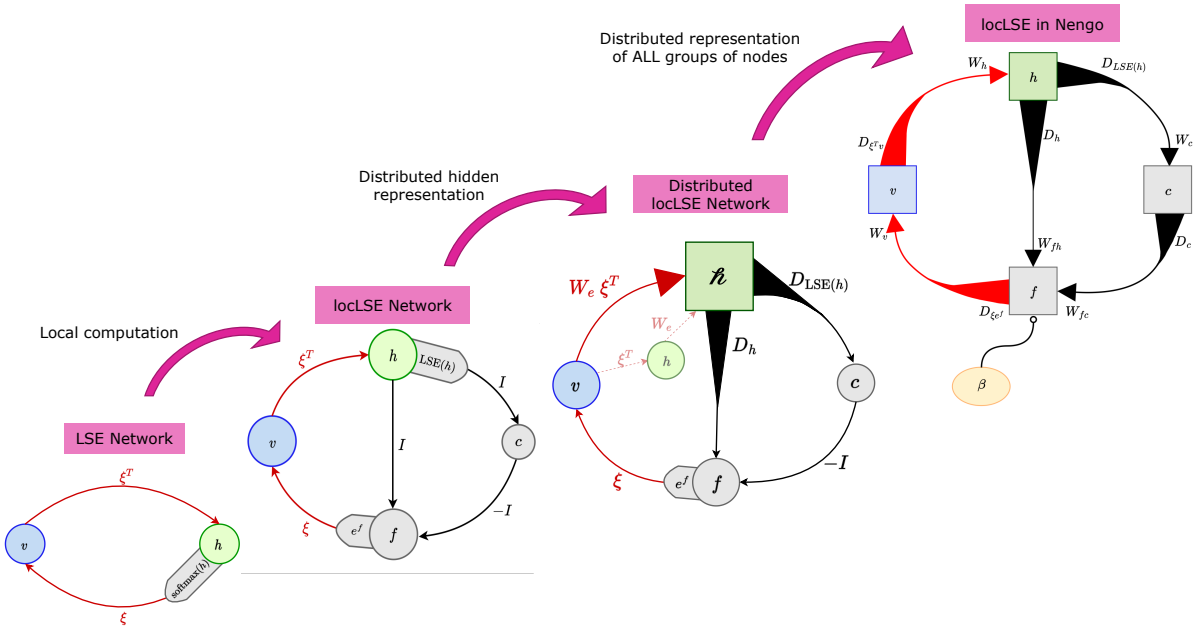


Figure 7.1: An outline of the steps made in this thesis to extend on the biological plausibility of the LSE network.

behaved almost exactly like the LSE network, converging to the target closest to the input at essentially the same rate for a variety of memory dataset sizes. Figure 5.1 shows the similarity in the dynamics of these two networks with both feature states promptly correcting certain bits from the input pattern so that the feature nodes converge to the target closest to it. The hidden dynamics are also identical here, with the largest value occurring at the index of the closest target. This one large value resulted in the softmax of $\boldsymbol{h}$ being mostly zeros with a value near 1 at the index of the closest target. Thus, we validated that the locLSE network behaves very similarly to the LSE in that it successfully acts as a CAM. In other words, the introduction of the local softmax sub-network does not disrupt the Hopfield dynamics.

Next, we created an even more biological version of the LSE network – the Distributed locLSE network – by applying the NEF to the hidden layer, and encoding the neurons into a higher dimensional distributed representation, $\boldsymbol{\hbar}$. Figure 4.1 shows that the Distributed locLSE network is created by projecting the hidden nodes $\boldsymbol{h}$ into $\boldsymbol{\hbar}$ – a higher dimensional

ensemble of neurons. To compute the correct dynamics, we learned decoding weights, $\boldsymbol{D_h}$ and $\boldsymbol{D}_{\text{LSE}(\boldsymbol{h})}$, which decode $\boldsymbol{h}$ and LSE($\boldsymbol{h}$) from $\boldsymbol{\hbar}$ respectively. The decoded values are then passed to the $\boldsymbol{f}$ and c nodes respectively. The connection weight matrix, $\boldsymbol{\xi}$ was also learned in this network. A test run on the Distributed locLSE network is shown in section 4.2, which uses the same memory dataset and input pattern as the test run for the locLSE network. In Figure 4.2 we see that the feature nodes, as well as the softmax sub-network nodes, $\boldsymbol{f}$ and c, behave just like those in the locLSE network in Figure 3.4. Specifically, the feature nodes quickly converge to the target closest to the input by correcting the 3 bits that are different. The hidden representation here looks very different from that of the LSE and locLSE networks; as opposed to relying on one of the $N_h = M$ hidden nodes to represent a target, the Distributed locLSE network has an ensemble of $N_{\hbar} = 2000$ neurons that collectively represent a target.

The results from the CAM experiments in Table 5.1 further verify that the Distributed locLSE network behaves a lot like the original LSE and the locLSE network. The table shows that the Distributed locLSE network converged to the target closest to the input at a rate very close to that of the other two networks for a variety of memory dataset sizes. Figure 5.1 shows an example from one of the CAM experiments where all networks converge to the closest target, showing the feature nodes of all 3 networks behaving very similarly. However, while the LSE and locLSE dynamics are essentially indistinguishable, the Distributed locLSE network's are slightly different. This is because of the distributed neural representation in the hidden layer. Recall that $\boldsymbol{h}$ and the LSE($\boldsymbol{h}$) are decoded out of $\boldsymbol{\hbar}$ to pass to the $\boldsymbol{f}$ and c nodes respectively. The decoding weights to get these values from $\boldsymbol{\hbar}$ are solved *approximately* for a set of samples. Hence, the decoded values of $\boldsymbol{h}$ and the LSE($\boldsymbol{h}$) will not be exact. These decoding errors are most likely what cause the differences in the dynamics. Despite this, the three networks converge to the closest target at very similar rates – the largest difference in these rates being 8%. It should be noted that no spurious states were observed in these experiments; the networks always converged to one of the trained targets. When the Distributed locLSE network converged to a target different than the closest, it almost always converged to the second closest target. Again, decoding errors might contribute to the HN going off in the wrong direction. It is also important to point out that the more target memories we want to store, the more neurons

are required to represent each target in the distributed hidden layer. More neurons in $\mathbb{h}$ allows for a higher-fidelity encoding and decoding. For larger memory datasets, more noise was also used in the training samples when solving for the decoding weights as this gave the bests results.

In the Distributed locLSE network, a collection of $N_{\mathbb{h}}$ hidden neurons collectively represent each target; this is much more biologically accurate than the LSE and locLSE networks which have $N_h = M$ hidden nodes, and rely on a single node per target. This is analogous to the "Grandmother" or "Jennifer Anniston" neuron theory which postulates that there is a single neuron in the brain that activates for certain concepts [14]. This one-to-one relationship between concepts and neurons is largely accepted to *not* be the case, and rather *many* neurons activate together to collectively encode and represent these concepts. Moreover, the Distributed locLSE network is more biologically realistic because it does not rely on a single neuron per target, and is thus robust to hidden neurons "dying". That is, similar to in human brains, if some neurons die, this doesn't necessarily cause an entire network to fail at recalling a concept. This behaviour is demonstrated in subsection 4.2.1, where we ran the Distributed locLSE network with 10% of its active hidden neurons dropped out, i.e. set to zero activity. We ran the LSE network with the same input and showed that both networks converge to the closest target. Figure 4.3 shows the similarity in dynamics between the LSE and Distributed locLSE network's feature nodes and the difference in dynamics between their hidden representations. The LSE network's hidden representation relies on the single node that represents the closest target to be large in comparison to the others in order to drive the feature nodes to match this closest target. One can imagine if we killed this one node, the hidden representation would then drive the feature nodes toward the target corresponding to the hidden node that is the next largest, failing to converge to the closest target. This is not the case for the Distributed locLSE network; since the hidden representation is more combinatorial, we can drop out 200 active distributed hidden neurons and the network still converges to the closest target. It is important to note here that the behaviour of a Distributed locLSE network with distributed hidden neurons dropped out would depend on *which* neurons are dropped. We made sure to drop active neurons, i.e., neurons that are not already zero. We chose randomly from these active neurons, but if we chose distributed hidden neurons that are slightly active

(closer to zero) the behaviour might be different from if we chose distributed hidden neurons that were more active (closer to 1). If we dropped a lot of neurons that are all highly active, the network might struggle to act as flawlessly. Future work could involve further investigation of the dropout experiment with more statistical analysis.

Finally, the results from implementing the locLSE network in Nengo show that the network can act as a CAM for smaller sized memory datasets ($M = 4, D = 10$) given a sufficient number of LIF-rate neurons in each ensemble. When we use LIF neurons the network acts as a less efficient CAM, converging to the nearest target 72% of the time as opposed to 90% with LIF-rate neurons. Using the same amount of LIF-rate neurons for a larger memory dataset ($M = 10, D = 10$) showed a decrease in the rate at which the locLSE network could successfully correct perturbations in input patterns (64%). However, when the number of neurons was increased by 2000 in each ensemble, this rate increased to 76% of the time. This suggests that the locLSE network in Nengo can work well as a CAM, we just need a *significant* amount of neurons in each ensemble to be able to handle larger memory datasets. The Nengo implementation could also be improved a lot by choosing the encoding weights more deliberately, but this is a project for future work.

It is worth noting that the performance in the CAM experiments were always improved by the local softmax sub-network. That is, the locLSE network performed better than the original LSE network in terms of acting as a CAM in every case (for the small and larger memory datasets, and when using both LIF-rate and LIF neuron types). It is interesting that the more biologically sound model, the locLSE network, behaves better as a CAM than the LSE in Nengo. One possible explanation is that the locLSE breaks up the softmax in a way that is easier for the distributed representations to compute. Instead of computing the entire softmax in one layer, it does so more effectively through several layers of neurons.

7.1 Relation to Other Fields

Why is modelling associative memory in a biologically accurate way important? First of all, investigating models of associative memory could aid in the progression of many different fields including neuroscience, medicine, and AI.

Associative learning and memory is a common form of information storage that humans and animals use for cognition on a daily basis. It has been claimed that understanding such processes should be the "foundation" to further our understanding of other forms of behavior and cognition in human and other animals [44]. A lot is known about how associative memory operates on a behavioural level, however, the cellular mechanisms responsible for the storage and retrieval of such associated signals remains unclear. In terms of the neurons and synapses involved in associative learning, it is presumed that activity-dependent plasticity is involved, i.e., Long-Term Potentiation (LTP) and Long-Term Depression (LTD) [36, 40]. This can be modelled using Hebbian Learning as is the case in HNs. It has also been suggested that the neurons in the brain responsible for the learning and storage of associative memories can be classified into two main groups; primary groups which "integrate exogenous signals in sensory cortices and innervate neurons in cognition and emotion brain areas", as well as secondary groups which "receive innervations from primary groups and integrate endogenous signals during cognitive processes" [41]. This supports the idea of having two main groups that drive associative memory, as is the case with the feature and hidden groups in the model studied in this work.

It is currently widely accepted that functions of memory, specifically associative memory, are carried out by structures in the temporal lobe, most notably in the hippocampus [23, 24, 37]. Krotov and Hopfield suggest that their model, the LSE network, could potentially model certain parts of the hippocampus. Specifically, they note the CA3 area, which involves substantial recurrent connectivity [30]. The CA3 area consists of a large population of pyramidal neurons, some of which are place cells. Place cells are neurons that fire when an animal is in a specific region of its environment, i.e. they exhibit spatially correlated firing patterns [25]. They suggest that one can think of the place cells as the hidden nodes in their model, which receive information from the feature nodes, similar to how place cells aggregate inputs from the grid cells and environmental features [20]. In fact, it has been shown that the CA3 area of the hippocampus has some involvement in associative memory functions such as pattern completion, forming new associations, and the retrieval of complete memories based on a partial memory [9]. Krotov and Hopfield also suggest that their model could also be related to the area CA1, which receives inputs directly from the entorhinal cortex, and projects back to it. Here, the CA1 pyramidal cells

would correspond to the hidden neurons and the cells in the entorhinal cortex would be the feature nodes [20]. However, they also indicate that the hippocampus, like most parts of the brain, is involved in many tasks and it is thus quite difficult to separate the network for associative memory from the networks required for other functions.

Associative memory abilities are also known to decline in neurodegenerative diseases such as Alzheimer's Disease [22, 26]. Furthering our understanding of how these memories are formed on a neurobiological level will aid in the development of therapeutics to combat the loss of associative memory abilities in people with neurodegenerative diseases. This knowledge could also help neuroscientists understand more about the causes of these diseases and make diagnosis easier.

Many other obstacles exist in the real world that can benefit from being approached through the lens of associative memory. For example, associative memory models are used to assist in AI tasks such as image processing (segmentation, feature extraction) [11], facial recognition [35], denoising information [3, 17], medical imaging tasks [7, 6], and other multiple instance learning (MIL) problems such as immune repertoire classification [45]. Thus, understanding more about biologically accurate neural models may help the advancement of AI. One key discrepancy between AI and the human brain is the amount of data needed for computers to perform certain tasks. For example, the IBM computer Watson required terabytes of reference material in order to beat human contestants on *Jeopardy!* [13]. The human brain is considerably less reliant on such amounts of data to perform tasks. Implementing systems that do not require such amounts of data would allow AI to advance in leaps and bounds.

Neuromorphic chips/computers are devices that are based on the brain in terms of function and hardware [33]. They consist of many small neuron-like circuits which govern their processing and memory. Examples of neuromorphic platforms include Intel's research chip *Loihi* [28], FPGA-based circuits [43], and others. These devices involve event-driven computation that is inherently parallel, allowing them to operate at much less power compared to traditional computing systems [33, 38]. Neuromorphic implementations of AI algorithms can sometimes have difficulty with non-local neural processes. Using biologically accurate algorithms, like the NEF, avoids this issue and enables speed ups [42]. Hence, as neuroscience and computer science progress, they each aid in the advancement

of the other.

7.2 Limitations and Future Work

Although the goal of this work was to further the biological plausibility of Krotov and Hopfield's modern HN, there still remain aspects that require more work and could be expanded on. Recall that we learned the connection weights, $\boldsymbol{\xi}$, using a local learning rule. However, a separate, symmetric set of connection weights, $\boldsymbol{\xi}^T$, are copied from the learned weights. One could implement a rule to learn the $\boldsymbol{\xi}^T$ connections, but this is left to future work. In fact, it is not strictly necessary for the two connection weight matrices to be the exact transposes of each other [16]. When we learn $\boldsymbol{\xi}$ for the distributed locLSE network, we clamp the hidden neurons to the distributed representation of each target we want to store. In doing so, we set $\boldsymbol{\hbar} = \sigma(\boldsymbol{W}_e(\boldsymbol{X}\boldsymbol{v}) + \boldsymbol{b}_e)$ and $d\boldsymbol{\hbar}/dt = 0$. That is, we encode the resulting $\boldsymbol{X}\boldsymbol{v}$ using a random encoding. Perhaps we could skip $\boldsymbol{X}$ altogether and simply randomly encode $\boldsymbol{v}$. This is also left for investigation in future work.

There are many directions that one could extend this work in. For example, the dropout experiment performed on the Distributed locLSE could be investigated further. For the purposes of this work, we did not perform any statistical analysis on how the distributed locLSE network behaves with dropped hidden neurons; we were mainly interested in simply illustrating that you *can* drop neurons and the network can still operate as it should. However, as mentioned in our discussion, the behaviour of the Distributed locLSE network with dropped hidden neurons would likely depend on *which* neurons are dropped. Thus, it may be worth quantifying the robustness of these representations to dropped neurons. Also, all experiments in our work deal with binary patterns. We did experiment with non-binary targets, however all networks were more robust to binary values. A deeper study into their behaviours on non-binary targets could be interesting future work.

It should also be noted that building networks in Nengo is very dependent on certain parameters - namely, those that construct the encoders and solve the decoders (i.e., `radius`, `eval_points`, etc). Such an implementation can thus be quite touchy/unstable, and so there remains considerable latitude for improving the Nengo model. Nengo has many built

in features that could help explore different biological aspects of the network. Learning the connection weights using the built in PES learning rule is something we had established as a next step, but unfortunately did not have time to successfully implement. Hence, we leave this for future investigations.

Chapter 8

Conclusion

Modern HNs are a type of Content Addressable Memory (CAM), meaning they can store a large number of memories and then retrieve them in their entirety when prompted by partial or perturbed versions. They have received considerable attention for their increased storage capacity, but also because they behave as a model of associative memory in humans. Despite being the basis of many cognitive processes, we still have only a rudimentary understanding of the neural mechanisms that underlie associative learning and memory.

In this work, we extended on the biological plausibility of the modern HN presented by Krotov and Hopfield. This model of associative memory is dubbed the LSE network here, a diagram of which can be seen in Figure 4.1(a). We proposed the locLSE network to overcome the non-local computations used in the LSE network. This issue was solved by incorporating the softmax sub-network to compute the softmax of neural activities using only local computations, as illustrated in Figure 4.1(b). We also derived a local learning rule to learn the connection weight matrix, $\boldsymbol{\xi}$, as opposed to setting them as the target memories directly.

However, the LSE and locLSE networks both rely on a single neuron in the hidden representation per target. If this one neuron were to become damaged or die, the network would no longer be able to recall the corresponding pattern. A more biologically realistic version would have a higher-dimensional, distributed hidden representation of each target. This idea inspired the Distributed locLSE network, illustrated in Figure 4.1(c), in which the

hidden layer has a more distributed representation of each target across a larger population of neurons. Instead of having $N_h = M$ neurons in the hidden layer, where each target relies on a single neuron, this network has a larger ensemble of N_h hidden neurons that collectively represent each target. Some of these hidden neurons can even be dropped out and the Distributed locLSE network still operates as it should, as seen in Figure 4.3. It was shown that both the locLSE and the Distributed locLSE networks behave as a CAM. This was verified using the CAM experiment which checks the rate at which a network (or a group of networks) converge to the target in memory closest to the input it was given. All 3 networks (the LSE, locLSE, and Distributed locLSE networks) have very similar performances in the CAM experiments for a variety of memory dataset sizes, shown in Table 5.1. This experiment successfully illustrated that using these biological constraints doesn't change how the network behaves in terms of being a CAM.

To apply population coding to all groups of nodes rather than just the hidden representation, we implemented the locLSE network in Nengo. It was shown that the locLSE network in Nengo can operate as a CAM for small memory datasets, provided there are a sufficient amount of neurons in each ensemble. The network struggles more with storing and retrieving targets from larger memory datasets, however it does better when more neurons are added to each ensemble. This implies that the locLSE network in Nengo can act as a CAM for large memory datasets, although a huge amount of neurons would be needed in each ensemble. It was also found that the breaking up the computations of the softmax into several layers caused the locLSE network in Nengo to perform better as a CAM compared to the original LSE in Nengo.

To some it may seem that studying biological plausibility poses an unnecessary restriction on the development of these models. However, it is crucial to bear in mind that a lot of advancement in AI has been made possible by thinking through the lens of neurobiology. Likewise, developments in neuroscience can be stimulated when approaching neurological mechanisms through the lens of mathematics and computation. To this end, the major goal of this work was less about making the biologically constrained models perform perfectly or better than the original, but more broadly about studying these networks in an interdisciplinary context, developing insights that can contribute to research in both AI and neuroscience.

References

[1] Bruno B Averbeck, Peter E Latham, and Alexandre Pouget. Neural correlations, population coding and computation. *Nature reviews neuroscience*, 7(5):358–366, 2006.

[2] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7:48, 2014.

[3] Ishan Bhatnagar and Shubhang Bhatnagar. QR code denoising using parallel Hopfield networks. *arXiv preprint arXiv:1812.01065*, 2018.

[4] Rafal Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of mathematical psychology*, 76:198–211, 2017.

[5] Rafal Bogacz and Kevin Gurney. The basal ganglia and cortex implement optimal decision making between alternative actions. *Neural computation*, 19(2):442–477, 2007.

[6] Chuan-Yu Chang. Contextual-based Hopfield neural network for medical image edge detection. *Optical Engineering*, 45(3):037006, 2006.

[7] Chuan Yu Chang and Pau-Choo Chung. Two-layer competitive based Hopfield neural network for medical image edge detection. *optical engineering*, 39(3):695–703, 2000.

[8] B. Coppin. *Artificial Intelligence Illuminated*. Jones and Bartlett illuminated series. Jones and Bartlett Publishers, 2004.

[9] Lorena Deuker, Christian F Doeller, Juergen Fell, and Nikolai Axmacher. Human neuroimaging studies on the hippocampal CA3 region–integrating evidence for pattern separation and completion. *Frontiers in cellular neuroscience*, 8:64, 2014.

[10] Marcia Earhard. Cued recall and free recall as a function of the number of items per cue. *Journal of Verbal Learning and Verbal Behavior*, 6(2):257–263, 1967.

[11] Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. Image processing with neural networks—a review. *Pattern recognition*, 35(10):2279–2301, 2002.

[12] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2003.

[13] David A Ferrucci. Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1–1, 2012.

[14] Charles G Gross. Genealogy of the "grandmother cell". *The Neuroscientist*, 8(5):512–518, 2002.

[15] D. O. Hebb. *The organization of behavior; a neuropsychological theory.* Wiley, 1949.

[16] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[17] Kwang Baek Kim and Doo Heon Song. Facial image denoising from degraded rough casual photographs using Hopfield neural network. *International Information Institute (Tokyo). Information*, 20(4A):2513–2518, 2017.

[18] Dwight J Kravitz, Kadharbatcha S Saleem, Chris I Baker, Leslie G Ungerleider, and Mortimer Mishkin. The ventral visual pathway: an expanded neural framework for the processing of object quality. *Trends in cognitive sciences*, 17(1):26–49, 2013.

[19] Dmitry Krotov and John Hopfield. Dense associative memory is robust to adversarial inputs. *Neural computation*, 30(12):3151–3167, 2018.

[20] Dmitry Krotov and John Hopfield. Large associative memory problem in neurobiology and machine learning. *arXiv preprint arXiv:2008.06996*, 2020.

[21] Dmitry Krotov and John J Hopfield. Dense associative memory for pattern recognition. *Advances in neural information processing systems*, 29:1172–1180, 2016.

[22] Andy CH Lee, Shibley Rahman, John R Hodges, Barbara J Sahakian, and Kim S Graham. Associative and recognition memory for novel objects in dementia: implications for diagnosis. *European Journal of Neuroscience*, 18(6):1660–1670, 2003.

[23] Jason Y Lee, Heechul Jun, Shogo Soma, Tomoaki Nakazono, Kaori Shiraiwa, Ananya Dasgupta, Tatsuki Nakagawa, Jiayun L Xie, Jasmine Chavez, Rodrigo Romo, et al. Dopamine facilitates associative memory encoding in the entorhinal cortex. *Nature*, 598(7880):321–326, 2021.

[24] Andrew Mayes, Daniela Montaldi, and Ellen Migo. Associative memory and the medial temporal lobes. *Trends in cognitive sciences*, 11(3):126–135, 2007.

[25] John O'Keefe and Jonathan Dostrovsky. The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat. *Brain research*, 1971.

[26] Mario A Parra, Sharon Abrahams, Robert H Logie, Luis G Méndez, Francisco Lopera, and Sergio Della Sala. Visual short-term memory binding deficits in familial Alzheimer's disease. *Brain*, 133(9):2702–2713, 2010.

[27] J Andrew Pruszynski and Joel Zylberberg. The language of the brain: real-world neural population codes. *Current opinion in neurobiology*, 58:30–36, 2019.

[28] Alpha Renner, Yulia Sandamirskaya, Friedrich Sommer, and E Paxon Frady. Sparse vector binding on spiking neuromorphic hardware using synaptic delays. In *Proceedings of the International Conference on Neuromorphic Systems 2022*, pages 1–5, 2022.

[29] Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, Colleen J Gillon, Danijar Hafner, Adam Kepecs, Nikolaus Kriegeskorte, Peter Latham, Grace W Lindsay, Kenneth D Miller, Richard Naud, Christopher C Pack,

Panayiota Poirazi, Pieter Roelfsema, João Sacramento, Andrew Saxe, Benjamin Scellier, Anna C Schapiro, Walter Senn, Greg Wayne, Daniel Yamins, Friedemann Zenke, Joel Zylberberg, Denis Therien, and Konrad P Kording. A deep learning framework for neuroscience. *Nature Neuroscience*, 22:1761–1770, 2019.

[30] Edmund T Rolls. The storage and recall of memories in the hippocampo-cortical system. *Cell and tissue research*, 373(3):577–604, 2018.

[31] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[32] Yoshio Sakurai. Population coding by cell assemblies—what it really is in the brain. *Neuroscience research*, 26(1):1–16, 1996.

[33] Catherine D Schuman, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, Bill Kay, et al. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2(1):10–19, 2022.

[34] Mallory A Snow and Jeff Orchard. Biological softmax: Demonstrated in modern Hopfield networks. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 44, 2022.

[35] Neha Soni, Enakshi Khular Sharma, and Amita Kapoor. Application of Hopfield neural network for facial image recognition. *IJRTE*, 8:3101–3105, 2019.

[36] Patric K Stanton and Terrence J Sejnowski. Associative long-term depression in the hippocampus induced by hebbian covariance. *Nature*, 339(6221):215–218, 1989.

[37] Wendy A Suzuki. Associative learning signals in the brain. *Progress in brain research*, 169:305–320, 2008.

[38] Chetan Singh Thakur, Jamal Lottier Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Wang, Elisabetta Chicca, Jennifer Olson Hasler, et al. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, 12:891, 2018.

[39] Endel Tulving and Zena Pearlstone. Availability versus accessibility of information in memory for words. *Journal of Verbal Learning and Verbal Behavior*, 5(4):381–391, 1966.

[40] Jin-Hui Wang and Shan Cui. Associative memory cells: formation, function and perspective. *F1000Research*, 6, 2017.

[41] Jin-Hui Wang and Shan Cui. Associative memory cells and their working principle in the brain. *F1000Research*, 7, 2018.

[42] Runchun Wang, Tara Julia Hamilton, Jonathan Tapson, and André van Schaik. A compact neural core for digital implementation of the neural engineering framework. In *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, pages 548–551. IEEE, 2014.

[43] Runchun M Wang, Chetan S Thakur, and Andre Van Schaik. An FPGA-based massively parallel neuromorphic cortex simulator. *Frontiers in neuroscience*, 12:213, 2018.

[44] Edward A Wasserman and Ralph R Miller. What's elementary about associative learning? *Annual review of psychology*, 48:573, 1997.

[45] Michael Widrich, Bernhard Schäfl, Milena Pavlović, Hubert Ramsauer, Lukas Gruber, Markus Holzleitner, Johannes Brandstetter, Geir Kjetil Sandve, Victor Greiff, Sepp Hochreiter, et al. Modern Hopfield networks and attention for immune repertoire classification. *Advances in Neural Information Processing Systems*, 33:18832–18845, 2020.