Towards Cognitively Plausible Associative Memories Capable of Learning Complex Tasks

Thaddé Rolon-Mérette

Thesis submitted to the University of Ottawa

in partial Fulfillment of the requirements for the

Doctorate in Philosophy Psychology

School of Psychology

Faculty of Social Sciences

University of Ottawa

Acknowledgements

I want to extend my deepest gratitude to my thesis committee, Dr. Denis Cousineau, Dr. Jean-Philippe Thivierge, Dr. Melanie Sekeres, and Dr. Mary Kelly, for their willingness to evaluate this thesis and invaluable insights. I am deeply grateful to my supervisor, Sylvain Chartier, whose outstanding guidance and patience have been a beacon throughout this journey. Our coffee meetings have enriched my work and brought joy to my academic pursuits—I will truly miss these moments. I extend my heartfelt thanks to my lab colleagues for their indispensable support and camaraderie. The fun times we shared have made daunting challenges far more bearable, and I carry forward fond memories as I wish them all the best in their future endeavours. Thank you so much, Nareg Berberian, Matt Ross, Kinsey Church and Matias Calderini. My sincere appreciation goes to my parents, Marcel Merette and Sueli Rolon dos Santos and my siblings Rita, Muriel and Diego. A sincere thanks to my brother Damiem Rolon-Merette, with whom I had the honour to share these special years. You have all given me unwavering support and have been my foundation through the ups and downs, and I could not wish for a better family. *Eu amo voces para sempre*. I would also like to thank my in-laws, Toni, Maguy, Ferdinand and Rodolphe Abou Hamad, for their acceptance and patience with me during all the difficult periods. I am truly grateful to call you family. Finally, I am profoundly thankful to my wife, Kristine Abou Hamad, who is the cornerstone of my life's work. This thesis would not have been possible without her boundless love and support. Her strength, love and encouragement have been my guiding lights, reminding me daily of the beauty and purpose of life.

Table of contents

Abstract

This thesis explores the development of artificial neural networks (ANNs) grounded in cognitive principles. Specifically, it focuses on Associative Memories (AMs) and how to improve their learning properties while retaining cognitive plausibility. To do so, gradual architectural and functioning changes are explored, ultimately leading to a framework for multilayered and multinetwork AMs capable of learning complex tasks while sharing the same underlying principle. Chapter 1 investigates the variability and consistency within an unsupervised AM in categorizing similar and distinct stimuli, demonstrating that while individual representations vary, the learning outcomes are consistent across networks. Chapter 2 combines unsupervised AMs to create a multilayered architecture capable of learning and gradually distinguishing highly correlated inputs. Chapter 3 integrates this unsupervised architecture into a Bidirectional Associative Memory (BAM) to form a multilayered BAM capable of learning non-linear tasks relying solely on local associative learning processes. Chapter 4 presents a novel binary transmission function for AMs that improves biological plausibility while maintaining high learning performance. It also investigates the new properties gained by binary encoding, such as information flow control and memory replay, which facilitate dynamic interactions between multiple networks. The thesis concludes by discussing the implications of these findings for artificial intelligence and cognitive modelling and suggests future research directions for developing more robust and adaptable neural network architectures. It emphasizes the growing importance of multidisciplinary collaboration across various perspectives and fields in unravelling the complexities of brain function.

Preface

It is increasingly clear that multidisciplinary collaboration between diverse perspectives and disciplines is critical to unravelling the complexities of brain function. As such, the motivation behind this thesis lies in the conviction that Artificial Neural Networks (ANNs) hold immense potential as cognitive models to explain brain function. ANNs that strive to align with cognitive and biological plausibility to study the intricacies of the brain can offer deeper insights into neurological and psychological models, thereby enhancing the understanding and prediction of human behaviour. The synergy between psychology and ANNs is reciprocal; the principles and mechanisms underlying brain function, when transposed into the realm of ANN, can serve as a conduit for developing models capable of tackling increasingly complex cognitive tasks. Therefore, this thesis is a confluence of these two worlds, striving to foster a deeper understanding of the brain through the lens of ANNs while using these insights to advance the field of ANNs itself. The work presented in this thesis was composed of the following peer-reviewed articles and conference proceedings:

Rolon-Mérette, T., Rolon-Mérette, D., & Chartier, S (2024). Are Zeros and Ones All You Need in a Recurrent Neural Associative Memory?. Cognitive Systems Research. (Manuscript in preparation).

Rolon-Merette, T., Rolon-Merette, D., & Chartier, S. (2023). Towards binary encoding in Bidirectional Associative Memories. Proceeding in The International FLAIRS Conference Proceedings (Vol. 36). Presented as a talk and poster at the International FLAIRS Conference Proceedings (Vol. 36). Won best short paper and poster award.

Rolon-Mérette, D., Rolon-Mérette, T., & Chartier, S. (2023). A Multilayered Bidirectional Associative Memory Model for Learning Nonlinear Tasks. *Neural Networks*., 167, 244-

265.

Rolon-Merette, T., Rolon-Merette, D., & Chartier, S. (2019). Different Brain, Same Prototype? Cognitive Variability Within a Recurrent Associative Memory. Proceeding in *2019 International Conference on Cognitive Modelling* (pp. 192-197).

Rolon-Mérette, T., Rolon-Merette, D., & Chartier, S. (2018). Generating Cognitive Context with Feature-Extracting Bidirectional Associative Memory. *Procedia computer science*, 145, 428-436. Paper also presented as a talk at the *2018 Annual International Conference on Biologically Inspired Cognitive Architectures (BICA)*, Prague, Czechia.

Rolon-Mérette, T., Rolon-Mérette, D., & Chartier, S. (2019). Génération de Contextes Cognitifs dans une Mémoire Associative Bidirectionnelle à Extraction de Caractéristiques. Abstract presented as a talk at the *41e congrès de la Société Québécoise pour la Recherche en Psychologie (SQRP)*, Mont-Tremblant, QC.

Rolon-Merette, T., Rolon-Merette, D., & Chartier, S. (2023). Apprentissage avec représentation binaire et bipolaire dans une Mémoire Associative. Abstract presented as a poster at the *45e congrès de la Société Québécoise pour la Recherche en Psychologie (SQRP)*, Sherbrooke, QC.

Rolon-Mérette, T., Rolon-Mérette, D., & Chartier, S. (2018). L‘ajout d’un contexte cognitif afin d’améliorer l’apprentissage dans un réseaux de neurones artificiels. Abstract presented as a poster at the *40e congrès de la Société Québécoise pour la Recherche en Psychologie* (SQRP), Québec, QC.

I. Introduction

The human brain is a complicated problem. While it enables a Ph.D. student to write a thesis, it is also the cause for the student's inevitable endless revisions. This is but a drop in the ocean of cognitive processes that arise from the parallel and distributed computations of the brain's vast network of interconnected neurons (Christophel et al., 2017; Goldstein, 2015). Yet, the precise mechanisms by which these networks assimilate and manage environmental stimuli to initiate behaviours are still not fully understood and continue to be the subject of ongoing research. In psychology, Artificial Neural Networks (ANNs) offer an exciting approach as they enable simulated experiments and testing of hypotheses without the need for invasive techniques (Cichy & Kaiser, 2019; Eliasmith, 2013; Hasson et al., 2020). ANNs are formal models inspired by biological neural networks' structure and function (Perconti & Plebe, 2020; Richards et al., 2019). However, many ANNs use processes that are not biologically plausible to model complex cognitive functioning. To overcome this problem, this thesis aims to develop a new ANN capable of learning complex cognitive tasks using principles grounded in cognitive processes occurring in the brain. By doing so, the hope is to take a step closer to creating a more truthful model of human cognition.

I. 1. Artificial Neural Networks

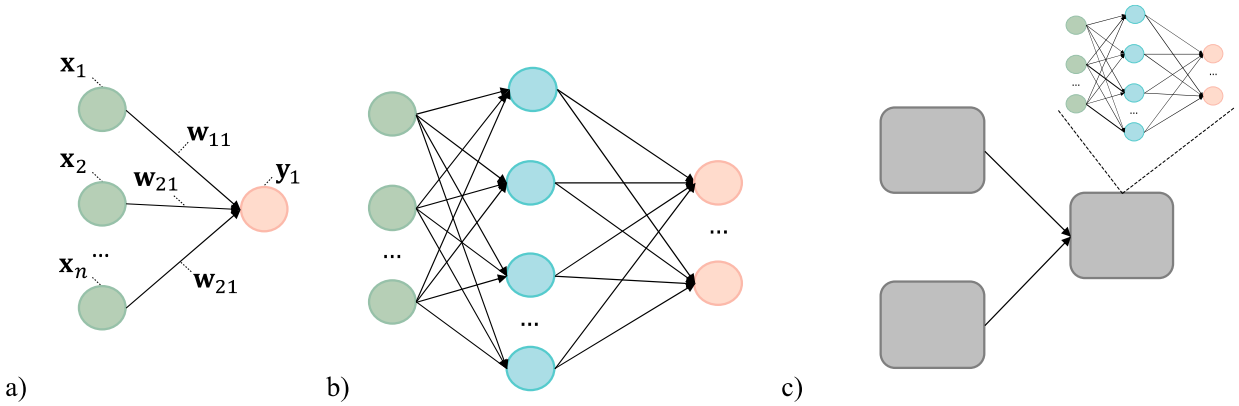

Fig. 1. Prototypical Artificial Neural Network architectures. (a) A two-layer architecture with inputs, **x**, represented by the green nodes in the input layer, connected by weights, **w**, to an output node (orange), **y**, in the output layer. (b) A multilayered architecture with a hidden layer (blue nodes) between the input and output layer. (c) A multi-network architecture where each grey square represents an individual network.

Typically, an ANN can be fully described by its architecture, learning rule and transmission function. The architecture refers to the network's topological structure, usually an arrangement of units, artificial neurons, and weights, the connections between units, into layers inspired by how neurons and synapses function in the brain. Prototypical ANN architectures are shown in Figure I. For example, an ANN might have a two-layer architecture with an input and output layer, as shown in Figure Ia, be multilayered with hidden layers between the input and output layers (Figure Ib) or even a multi-network architecture which is composed of many interacting individual networks (Figure Ic). The first units (green circles in Figure I) usually represent the input, denoted as vector **x**, while the units in the final layer represent the network output, denoted as vector **y**. These units are interconnected by weights (black arrows) with the entire weights between two layers usually referred to as a weight matrix, here **W**. Units in the hidden layers between the input and output layer are called hidden units.

ANNs are computational programs. However, their learning ability distinguishes them from other computational models (Macpherson et al., 2021). Initially, an ANN has no inherent capabilities as its weights are randomly set. However, through a learning process, an ANN can find appropriate weights to solve a specific task. This weight tuning is done according to a learning rule, which usually minimizes some error between the network's output and the desired result (Haykin, 1998). The learning rule, also known as the optimizer, is crucial in determining how connections between artificial neurons are changed to learn new stimuli. The most effective learning rules in ANNs, such as backpropagation, do not always reflect the functioning of biological neurons, making their explanatory power regarding human learning questionable (Cichy & Kaiser, 2019). On the other hand, learning rules based on Hebbian principles, reformulated by the adage "neurons that fire together wire together," are closer to actual neurons that operate strictly through local communications (Lim, 2021). In addition, an ANN also consists of a transmission function. The transmission function, also referred to as the activation function, determines how inputs are transformed into outputs as they pass through the network. It is often, but not always, a nonlinear function applied to each unit's summed input (Sharma et al., 2017). For example, the sigmoid function (Figure IIa) can be used to learn stimuli in binary form (0, 1), the tanh function (Figure IIb) can be used to learn stimuli in the bipolar form (-1,1), and the ReLU is a common function (Figure IIc) used in more engineering-based settings. The choice of transmission function can vary greatly depending on the network type, learning rule, architecture, or the desired encoding space.

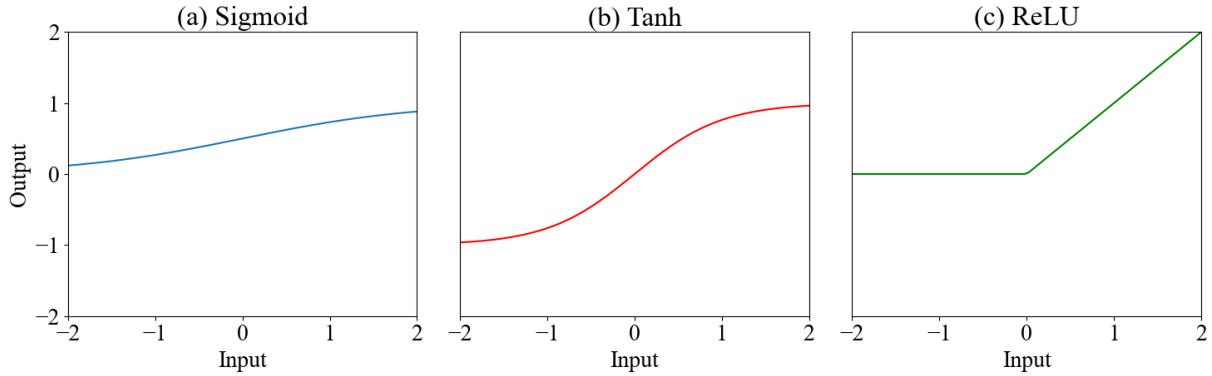

Fig. II Typical transmission functions with the (a) sigmoid, (b) Tanh and (c) ReLU functions shown.

ANNs have been around for several decades. The Perceptron introduced by Rosenblatt (1958) was the first ANN encompassing an architecture, transmission, and learning capabilities, paving the way for subsequent models of associative memory. In recent years, there has been a resurgence of interest in ANNs, mainly due to advancements in hardware technology (Aggarwal, 2018). Today, various ANNs exist, each with its architecture, learning rule, and transmission function (Leijnen & Veen, 2020). In psychology, ANNs have been used in a multitude of ways, from advanced data analysis tools to computational models of various cognitive and biological processes (Shao & Shen, 2023; Urban & Gates, 2021; Yang & Wang, 2020). Thus, an important distinction must be made between ANNs developed for practical applications and those as models of the brain (Blazek & Lin, 2021; Kriete & Noelle, 2005; LeCun et al., 2015; Taherkhani et al., 2020). For example, many ANNs in practical applications use the Backpropagation (BP) learning rule which has been long debated to be at odds with biology for many reasons, such as the backpropagation of information and the presence of global error (Lillicrap et al., 2020; Mazzoni et al., 1991; O'Reilly, 1996; Rumelhart et al., 2013; Saxe et al., 2021; Song et al., 2020; Stork, 1989). Therefore, for ANNs to be good models of cognition, they must aim to respect the properties of real neural networks (Grossberg, 1987; O'Reilly, 1998).

I. 2. Cognitive Plausibility

There are different approaches to implementing biological realism in ANNs, as there is often a trade-off between physical realism and computational proficiency. One method is to closely replicate the specific properties and behaviour of individual neurons in the network, such as in the case of spiking neural networks (Taherkhani et al., 2020). Such an approach may lead to a high degree of biological fidelity but can be computationally expensive and typically limited to small networks and simple tasks (Beniaguev et al., 2021; Markram et al., 2015). Another view is to adopt a more abstract understanding of biology, focusing on the more extensive and distributed networks, leading to ANNs capable of more complex cognitive functions (Pulvermüller et al., 2021; Yang & Molano-Mazón, 2021). This approach does necessarily not replicate the details of individual neurons but instead applies biological constraints to the dynamics of the network (O'Reilly, 1998). For instance, each unit in the network does not necessarily represent a single neuron, but a population of neurons, meaning each weight represents the combined signal from such neurons. This proposal adopts such a strategy and aims to be grounded in the group neurodynamic of neural networks and less in individual neurons while avoiding methods that simply increase computational efficiency (Harnad, 1990; Haykin, 2009).

I. 3. Bidirectional Associative Memories

A notable group of ANNs used in cognitive modelling is Associative Memories (AMs). Such systems aim to model associative learning (Anderson & Bower, 1972, 2014). They operate under the general idea that stimuli are encoded using weighted connections and can then be recalled based on the similarity between the input shown and the content of the stimuli stored in memory. The first AMs were proposed decades ago (Anderson & Bower, 1972; Kohonen, 1974;

Steinbuch, 1961; Willshaw et al., 1969). These models laid the groundwork for the concept of memory in computing based on the principles of human cognition. However, they could only handle orthogonal patterns. AMs started getting significant attention in the later 70s and early 80s with the introduction of feedback neurodynamics by Anderson et al. (1977) and the conception of attractors by Hopfield (1982) which introduced the concept of energy landscapes in memory models. Specifically, the Hopfield Network used recurrent connections and the Hebbian learning rule, a mechanism to update weights according to co-occurrences in the environment based on the classic postulate by Hebb (1949) By doing so, the network only uses the locally available information to form stable attractors for each input, allowing the model to perform auto-associations consisting of a stimulus associated with itself (Lim, 2021). However, the Hopfield network could not learn hetero-associations, where two different stimuli are associated. This limitation was overcome with the introduction of the Bidirectional Associative Memory (BAM) by Kosko (1988). BAMs use feedback connections from the output to the input layer, making the system noise-tolerant and able to recall stimuli using partial information (Barra et al., 2023; Chartier & Boukadoum, 2006a, 2011). Since then, many BAMs have been developed with exciting properties, see Acevedo-Mosqueda et al. (2013) for a review on BAMs. Many BAMs also use a Hebbian learning rule that is Hebbian based, although usually some form of error is added in the equation. It is also noteworthy that many other BAMs have diverged from the Hebbian learning constraint and have been developed to be used in engineering applications and not cognitive modelling.

I. 4. Problematic and related work

The standard BAM model has shown great potential. Still, it is quite limited in the complexity of tasks it can learn and therefore the extent of cognitive processes it can model. For instance, the BAM is unable to handle nonlinear associations. A typical non-linear problem is the XOR task which has a cognitive equivalent of understanding that a cat may like to be petted or played with but not both or none. There are nonlinear associations that must be made between pet/play and "good" and both/none and "bad." Although it is a simple nonlinear task, the BAM alone could not solve this problem. Of course, humans can learn these associations even at a very young age and are constantly exposed to both linear and nonlinear problems throughout their life (Levering et al., 2020). In fact, many cognitive tasks typically used in Psychology require learning nonlinear associations (Rosedahl & Ashby, 2022). As such, several methods have been proposed to enable the BAM to learn more complex tasks, like non-linear ones. For example, the alpha-beta BAMs and morphological BAMs use lattice (minimax) algebra and can deal with non-linearity (Díaz de León & Gamino Carranza, 2022; Ritter et al., 1999; Yáñez-Márquez et al., 2018). Some can also exhibit other interesting properties, such as the ability of one-shot learning, where associations are seen to be learned after a single exposure (Lee et al., 2015). However, these models branched away from the traditional view of the artificial neuron (Anderson et al., 1977; McCulloch & Pitts, 1943) at the cost of increasing the degree of complexity of the base unit of the model.

Another method is to modify the BAMs architecture, i.e., changing how artificial neurons are interconnected. For instance, the standard BAM has a basic 2-layer architecture, while most ANNs capable of learning non-linear tasks and other complex cognitive tasks often use a multilayer architecture. This is because information processing through multiple layers seems to transform nonlinear information into linearly separable representations. Usually, a change in

architecture is done in conjunction with a change in internal functions. Recent studies have shown that using a multilayered architecture allows the BAM to learn nonlinear tasks (Adigun & Kosko, 2019; Kosko, 2021; Singh & Saraswat, 2017). However, success in these studies could be attributed to the Backpropagation learning rule, which has been previously stated to be problematic for cognitive plausibility, or require external interventions, like preprocessing task stimuli to provide an explicit description of the rules. Still, they show that a multilayered architecture could enable the BAM to learn more complex tasks.

Other recent methods suggest the interaction of multiple ANNs with different architectures to learn complex tasks. For instance, a multilayered network could be used prior to the BAM to pre-process non-linear inputs into linearly separable ones (Liu et al., 2019; Wu et al., 2018). Other approaches used many ANNs, each with different architectures and internal functions (Hersche et al., 2023; Masse et al., 2018). However, most of these methods still rely on Backprogagation. Furthermore, these approaches use networks with divergent internal dynamics, which does not correspond to what we know of human cognition. In reality, although the human brain is composed of networks with varied functions, all these networks operate with similar internal neurodynamics (O'Reilly, 1998). Therefore, for this approach to offer a plausible explanation of rule learning in humans, the various layers and networks must operate by having the same internal functions. Another interesting approach is utilizing a localist representation, such as in the Adaptive Resonance Theory (ART), where a model assigns each new memory to a distinct unit (Carpenter & Grossberg, 1988; Grossberg, 2013). However, the goal here is to achieve this same result from a distributed neurodynamic perspective.

I. 5. Proposed Solution

A potential solution could be found in the Feature Extracting Bidirectional Associative Memory (FEBAM), an unsupervised variant of the BAM (Chartier et al., 2007). Because it is unsupervised, it generates representations of inputs through features extraction. These representations can be used as input for the BAM, a combination that has successfully solved some simple nonlinear problems, such as XOR (Chartier et al., 2012; Morissette & Chartier, 2013; Tremblay et al., 2013). However, these results were inconsistent, only working a fraction of the time and not generalizing to more difficult nonlinear tasks. Additionally, the FEBAM's generated representations can be used as contextual information to help the BAM perform multiple tasks (Church et al., 2020; D. Rolon-Mérette et al., 2019; T. Rolon-Mérette et al., 2019), but these studies were preliminary and did not extend beyond simple associations. Despite these shortcomings, using the FEBAM as a foundation to form a more extensive architecture remains a promising approach. This thesis proposes that employing the FEBAM as separate layers in constructing a more comprehensive architecture will enable the system to handle nonlinear tasks effectively. This network design will consistently apply the same internal mechanisms across its entire structure. Thus, any behavioral alterations can be attributed directly to architectural modifications rather than external influences or additional complexities. As this thesis progresses, the focus will be on progressively developing larger networks, specifically by incorporating multiple layers. In the final chapter, we will refine the models' internal dynamics by introducing a novel transmission function. This adjustment aims to allow binary encoding, which, among other features, will support the implementation of parallel networks.

I. 6. Overview of Chapters

The four chapters of this thesis were written for publication at the University of Ottawa under the guidance of Professor Sylvain Chartier. Each chapter offers unique methodological contributions while sharing thematic connections and gradual progression. A central focus of the thesis is the development and combination of BAM and FEBAM models in creating new architectures with consistent internal dynamics to learn more complex tasks.

Chapter One delves into the impact of altering the number of units in a single FEBAM, utilizing categorization tasks to examine prototypes recall with fewer units and exemplars recall with a higher number of units. An important finding was the consistent behaviour across different networks and task conditions, reinforcing its suitability as a component in multilayered networks. This chapter was published as a conference proceeding and presented at the 2019 International Conference on Cognitive Modelling (ICCM). The project was financially supported by the Ontario Graduate Scholarship (OGS) and the Natural Sciences and Engineering Research Council of Canada (NSERC). Thaddé Rolon-Mérette conceptualized the study, designed and implemented the methodology in Python and ran the numerical simulations. Thaddé Rolon-Mérette, Damiem Rolon-Mérette, and Matias Calderini analyzed the results and wrote the article. Sylvain Chartier reviewed and edited the article.

The Second Chapter created a multilayered FEBAM architecture. The goal was to examine the generated representations across varying layers. A key finding was that representations become progressively less correlated as they are generated through the deepest FEBAM layers. This finding was pivotal to show that complex and highly correlated inputs would become more separated in space by using multiple layers. And this was done without sacrificing cognitive plausibility. This chapter was published as a journal article in *Procedia Computer Science*, also

with NSERC funding. Thaddé Rolon-Mérette conceptualized the study, designed and implemented the methodology in Python, ran the numerical simulations and analyzed the results. Thaddé Rolon-Merette and Damiem Rolon-Merette wrote the article. Sylvain Chartier reviewed and edited the article.

In Chapter Three, the multilayered FEBAM architecture was integrated with a BAM to form a new model called the Multi-Feature Extracting Bidirectional Associative Memory (or MF-BAM). The goal was to use the decorrelated generated representations of a multilayered FEBAM to allow the BAM to learn non-linear tasks. Findings confirm this, as the MF-BAM learned numerous nonlinear tasks. Once again, the cognitive properties of the BAM and FEBAM were not compromised to achieve this. In fact, the creation of linearly separable representations was an unintended consequence of the associative processes within the model's layers. This chapter was published as a journal article in *Neural Networks*, supported by NSERC, the Ontario Graduate Scholarship (OGS), The Fonds de recherche du Québec (FQRNT) and NSERC. Damiem Rolon-Merette, Thaddé Rolon-Mérette and Sylvain Chartier conceptualized the study. Damiem Rolon-Merette and Thaddé Rolon-Merette designed and implemented the methodology in Python, ran the simulations, analyzed the results and wrote the article. Sylvain Chartier reviewed and edited the article.

The Fourth chapter introduces a new transmission function enabling binary encoding, which has cognitive benefits and offers better biological plausibility, while retaining the learning performance and properties observed in standard bipolar BAMs. The chapter provides a detailed explanation the new transmission function and explores several improvements compared to the standard bipolar transmission function. Part of this chapter was published as a conference proceeding and presented in the 2023 international conference of Florida Artificial Intelligence

Society (FLAIRs-23) for which it won the best poster award. The entire chapter is slated for submission to a Q1 journal in early 2024, with OGS and NSERC funding. Thaddé Rolon-Mérette conceptualized the study. Thaddé Rolon-Mérette and Sylvain Chartier developed the binary transmission function and designed and implemented the methodology in Python. Thaddé Rolon-Mérette and Damiem Rolon-Merette ran the computer simulations and analyzed the results. Sylvain Chartier reviewed and edited the article.

Finally, this thesis ends with a brief discussion to contextualize the findings within the broader landscape of cognitive modelling and neural network research. It reflects on the implications of these results for future studies in artificial intelligence and psychology and proposes potential avenues for applying these models to real-world problems. The discussion also highlights the limitations of the current work and suggests directions for further research to refine and expand upon the developed architectures and methodologies.

1. Chapter I

Different Brain, Same Prototype? Cognitive Variability within a Recurrent Associative Memory

Abstract

When learning similar stimuli, we tend to group them together. This categorization is a behaviour which all humans share. Yet, the pathways undertaken by the brain differs between individuals. To investigate this phenomenon, a Feature Extracting Bidirectional Associative Memory (FEBAM) was used to generate representations of various grouped stimuli. It was determined that representations created by different FEBAMs were always new. However, the learning behaviour was always the same. The generated representations were always categorized into the right category. Finally, by lowering the size of these representations, prototypes of the categories could be created. Recall tests showed that reconstructed prototypes remained the same across all FEBAMs, even if the representations themselves differed. This shows that although the encoding pathways might differ between individuals, the learned cognitive concepts do not. These findings are promising steps towards better understanding how individuals exhibit common cognitive functionality despite variability in neural activity.

Keywords: *Variability; Categorization; Feature-Extraction; Associative Learning; Bidirectional Recurrent Neural Networks, Cognition.*

1.1 Introduction

There are billions of human beings on this planet and each one of them can understand and share complex concepts such as language, games, music and much more. This common cognitive understanding is mind boggling when individuality is considered. There is no brain that is the same as another, with each containing a unique arrangement of its neural structures and connections (Genon et al., 2022; Sporns et al., 2005; Thompson et al., 1996). When presented with the same learning task, different individuals will exhibit different neural activity (Churchland et al., 2010; Mueller et al., 2013). While perception can change based on certain differences in anatomical structures, surprisingly, this variability does not seem to drastically change an individual's understanding of the world and/or its relationships with others. While the neurological pathways involved are different across individuals, the behaviour remains consistent. In other words, from different neural activities, the same cognitive functionality can be observed. That being said, the mechanisms behind such commonality from variability are yet to be fully understood.

An encouraging avenue to better understand this would be to explore the concept of associative learning and categorization. Associative learning can be seen as linking two or more stimuli together (Anderson & Bower, 2014; Rescorla, 1972). One of its interesting characteristics is the ability to recall one stimulus when only presented with a partial cue (McClelland et al., 1995). This process is believed to form the basis of categorization whereas similar patterns are grouped together to form a category (Hampton, 1997; Shields & Rovee-Collier, 1992; Wasserman et al., 2015). However, how these "grouped" patterns are represented in our brain remains a mystery. Are the encoded representations of stimuli different across individuals? If so, do they respect the relationship between stimuli, i.e. correctly categorized? In other words, if stimuli are similar, are their representations also similar?

In cognition, such questions can be explored using formal models (Forstmann et al., 2011). Specifically, artificial neural networks (ANNs) have been an exciting approach to study various key cognitive concepts such as associative learning and categorization (Mareschal et al., 2000). One of the many interesting properties of ANNs dwells in the initialization of weight connections. By randomly initializing the connection weights, each individual instance of a network will be different, analogous to the variability found in human brains. However, what would be interesting is that different instances of a network would display the same behaviour when presented with the same learning task.

Among ANNs are Recurrent Associative Memories, which are designed to implement associative learning (Acevedo-Mosqueda et al., 2013). Particularly, there is the Feature Extracting Bidirectional Associative Memory, or FEBAM (Chartier et al., 2007), which can create perceptual features from input patterns via feature extraction (Rolon-Mérette et al., 2018). This property allows the FEBAM of category development (grouping similar patterns together based on their correlation). However, a question remains. When presented with the same stimuli, will the FEBAM always generate new representations and if so, will it exhibit the same learning behaviour? In other words, will the representations be categorized in the same manner even if they are always new? This would shed light on the mechanisms allowing common cognitive functionality found between individuals.

The next section gives a short description of the FEBAM and a cluster analysis, followed by three simulations. In simulation I, it was investigated if the representations created by different instances of the FEBAM are always new. In simulation II, the exemplar categorization was observed with a learning task consisting of grouped patterns. In simulation III, under the same

learning task, the size of representations was varied to examine prototype categorization. Finally, this paper ends with a short discussion.

1.2 Model

The FEBAM is a completely unsupervised recurrent ANN, meaning it does not have any explicit teacher. The entirety of the model can be described by its architecture, transmission function and learning function.

1.2.1 Architecture

The FEBAM architecture is illustrated in Fig. 1.1. The model has two layers of interconnected units in a bidirectional fashion, where the **W** and **V** layers return information to each other. Contrary to traditional bidirectional associative memories, there is only one explicit connection, $\mathbf{x}_{[0]}$, to allow the network to perform feature extraction. The number of units in the y-layer (u) can thus be manipulated.

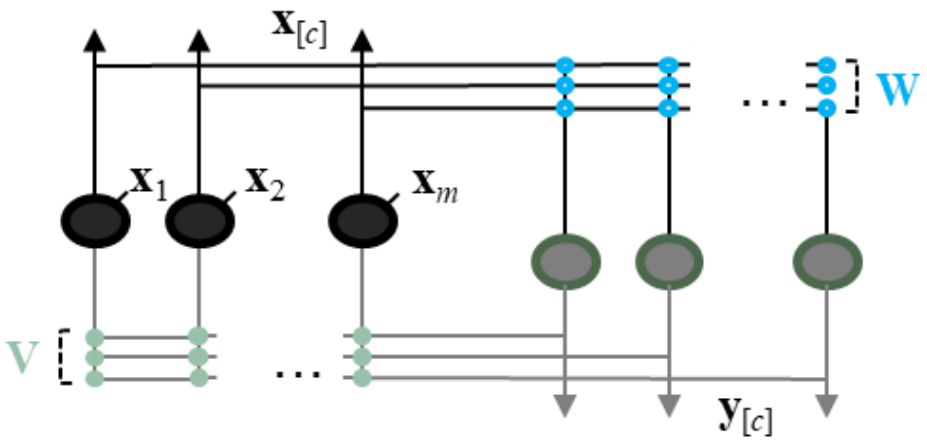

Fig 1.1 Architecture of the FEBAM

1.2.2 Transmission function

The transmission function is defined by the Equations 1.1 and 1.2:

$$(1.1)\ \forall i, \dots, n, \mathbf{y}_{i[c+1]} = f(\mathbf{a}_{i[c]}) = \begin{cases} 1, if\ \mathbf{a}_{i[c]} > 1 \\ -1, if\ \mathbf{a}_{i[c]} < -1 \\ (\delta + 1)\mathbf{a}_{i[c]} - \delta \mathbf{a}_{i[c]}^3, Else \end{cases}$$

$$(1.2)\ \forall i, \dots, m, \mathbf{x}_{i[c+1]} = f\left(\mathbf{b}_{i[c]}\right) = \begin{cases} 1, if\ \mathbf{b}_{i[c]} > 1 \\ -1, if\ \mathbf{b}_{i[c]} < -1 \\ (\delta + 1)\mathbf{b}_{i[c]} - \delta \mathbf{b}_{i[c]}^3, Else \end{cases}$$

Where n and m are the total number of units in each layer, i is the index unit, c is the iteration cycles index, δ is the general transmission parameter and **a** and **b** are the activations. These activations are obtained the following way: $\mathbf{a}_{[c]}=\mathbf{W}\mathbf{x}_{[c]}$ and $\mathbf{b}_{[c]}=\mathbf{V}\mathbf{y}_{[c]}$.

1.2.3 Learning process rule

The weights are modified following a hebbian/anti-hebbian rule following equations 1.3 ad 1.4:

$$(1.3)\ \ \mathbf{W}_{[k+1]} = \mathbf{W}_{[k]} + \eta\left(\mathbf{y}_{[0]} - \mathbf{y}_{[c]}\right)\left(\mathbf{x}_{[0]} + \mathbf{x}_{[c]}\right)^{\mathrm{T}}$$

$$(1.4)\ \ \mathbf{V}_{[k+1]} = \mathbf{V}_{[k]} + \eta\left(\mathbf{x}_{[0]} - \mathbf{x}_{[c]}\right)\left(\mathbf{y}_{[0]} + \mathbf{y}_{[c]}\right)^{\mathrm{T}}$$

Where $\mathbf{x}_{[0]}$ is the initial input, $\mathbf{y}_{[0]}$ is the generated representation, $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$ are their respective reconstructions after c cycles, η is the learning parameter and k is a given learning trial. Equation 2a and 2b shows that the matrix weights will converge when $\mathbf{x}_{[0]} = \mathbf{x}_{[c]}$ or $\mathbf{y}_{[0]} = \mathbf{y}_{[c]}$. To reduce the simulation time the number of iterations was set to $t = 1$. It is guaranteed that the learning will converge if the learning parameter (η) is smaller than the value in equation 1.5 (Chartier & Boukadoum, 2006a):

$$(1.5)\ \eta\ < \frac{1}{2(1-2\delta)\mathrm{Max}[m,n]}, \delta\ \neq\ \frac{1}{2}$$

1.2.4. Iteration cycle

As previously mentioned, in the FEBAM, there is only one explicit connection $\mathbf{x}_{[0]}$, meaning the $\mathbf{y}_{[0]}$ inputs are not initially available. Instead, they are obtained after a first iteration through the network. As shown in Fig. 1.2, $\mathbf{y}_{[0]}$ is obtained by the iteration of $\mathbf{x}_{[0]}$ through its corresponding weight connections $\mathbf{W}$ using the transmission function. Subsequently, $\mathbf{x}_{[1]}$ is obtained from $\mathbf{y}_{[0]}$ and finally, $\mathbf{y}_{[1]}$ from $\mathbf{x}_{[1]}$. Through the weight updates, each $\mathbf{x}_{[1]}$ and $\mathbf{y}_{[1]}$ will converge to a solution that will try to best reconstruct its associated initial pattern $\mathbf{x}_{[0]}$ or its initial generated representation $\mathbf{y}_{[0]}$.Thus, in the case where $\mathbf{x}_{10]}$ does not equal $\mathbf{x}_{[0]}$, weight convergence will be granted by $\mathbf{y}_{[0]} = \mathbf{y}_{[1]}$. The number of units in the $\mathbf{y}$–layer (u) determined the dimensionality (size) of the generated representation.

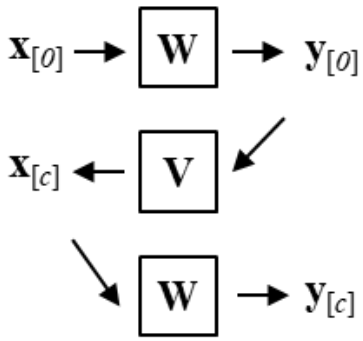

Fig 1.2 Iterative process for obtaining all components for weight updates in the FEBAM. Initially, $\mathbf{x}_{[0]}$, the set of external inputs, is processed by the weight matrix $\mathbf{W}$, which in turn generates the representations $\mathbf{y}_{[0]}$. These representations are then passed through another weight matrix $\mathbf{V}$, producing $\mathbf{x}_{[1]}$. Subsequently, $\mathbf{x}_{[1]}$ is fed back through $\mathbf{W}$, resulting in $\mathbf{y}_{[1]}$. This iterative cycle is repeated for c cycles, ultimately yielding the final outputs $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$.

1.2.5. Learning Procedure

During learning, the transmission function's parameter (δ) was set to 0.2 and the learning parameter (η) respected Equation 3 for all the simulations. Weights were randomly initialized with values between -0.1 and 0.1. Learning stopped when the network achieved a means squared error

(MSE) of less than 10^-10 or when 5000 learning trials was reached. Learning was conducted following this procedure:

1.Creation of a list of inputs respecting preset conditions.

2.Random selection of a given exemplar from the list to obtain $\mathbf{x}_{[0]}$.

3.Iteration through the network (as illustrated in Figure 2) using the output function to obtain $\mathbf{y}_{[0]}$, $\mathbf{x}_{[1]}$ and $\mathbf{y}_{[1]}$.

4.Computation of weight updates according to the learning rule.

5.Repetition of steps 2) to 4) until the minimum mean squared error between $\mathbf{x}_{[0]}$ and $\mathbf{x}_{[1]}$ or $\mathbf{y}_{[0]}$ and $\mathbf{y}_{[1]}$ or maximum trials is reached.

1.3 Cluster Analysis

In order to partition the generated representations into categories, k-mean clustering was used. For a chosen number of clusters *k*, the algorithm randomly sets *k* centroids in feature space and assigns each data point to the category of its nearest centroid. The positions of each centroid are then iteratively readjusted such that the within-category distance of the resulting categories is minimized. Lloyds algorithm and K-means++ initialization were implemented with the SickitLearn library on Python (Arthur & Vassilvitskii, 2006; Kanungo et al., 2002). The sum of the squared distances between data points and their centroid is presented by distortion. A priori, the number of clusters that would most appropriately divide the data cannot be known and its high dimensionality makes it prohibitive to determine it visually. Instead, the elbow method was applied to select the optimal number of clusters (Kodinariya & Makwana, 2013). Cluster analysis was conducted under two different scenarios. Scenario A will be used to examine variability across all

FEBAMs (Simulation I and IIIb). Scenario B allows to find the average behaviour of an individual FEBAM (Simulation II and IIIa).

1.3.1 Scenario A

1. Creation of input patterns respecting preset conditions.
2. FEBAM learning as specified in the learning procedure.
3. Repetition of steps 1) and 2) for all FEBAMs.
4. K-Means cluster analysis on generated representations of all FEBAMs at once from step 3).

1.3.2 Scenario B

1. Creation of input patterns respecting preset conditions.
2. FEBAM learning as specified in the learning procedure.
3. K-Means cluster analysis on generated representations of each individual FEBAM.
4. Repetition of steps 1) to 3) for all FEBAMs.
5. Calculate the average distortion and number of clusters from step 4).

1.4. Simulation I: New representations

The number of generated features was studied when the inputs were kept constant. The task consisted of three learning conditions of different input patterns and generating their associated representations. In each condition, the patterns were fed to multiple FEBAMs, mimicking the learning process of different individuals. The generated outputs, or representations, were then analyzed with k-means clustering using Scenario A.

1.4.1 Methodology

Three different learning conditions were studied using pixelated bipolar inputs, where black pixels represent the value of +1 and white pixels -1. The “single” condition consisted of a single pattern. The “category” condition comprised two categories of five highly correlated patterns. Each pattern within a category exhibited a correlation of 0.95 and the correlation between patterns of both categories was set to 0.15. Finally, in the “random” condition, ten inputs were generated with low correlations varying from 0.01 to 0.30. All three conditions are presented in Fig. 1.3.

In order to have a good estimate of the behaviour, the input patterns were presented to 1000 different FEBAMs, each with a different set of randomly initialized weight connections. The size of the generated representations was kept constant at a dimension of 50 ($\underline{u = 50}$). Finally, for each condition, k-means clustering analysis was conducted on the generated representations of all the FEBAMs at once, as stated in Scenario A.

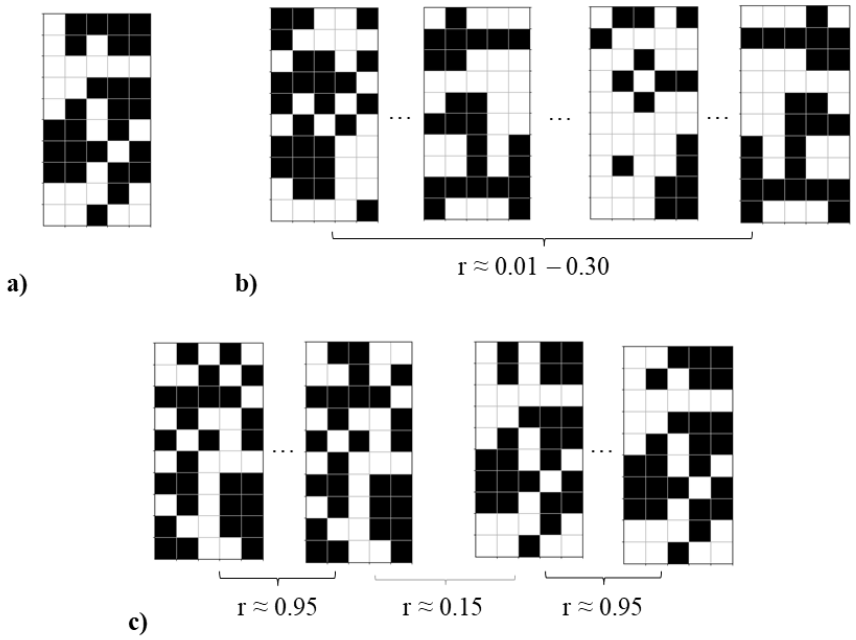

Fig 1.3 Input patterns for the “single” (a), “random” (b) and the “category” (c) conditions. The correlation (r) between patterns is shown.

1.4.2 Results

Fig. 1.4 shows the results of k-means clustering for each condition. As the number of clusters created increased, the distortion decreased. However, for all three conditions no elbow was observed.

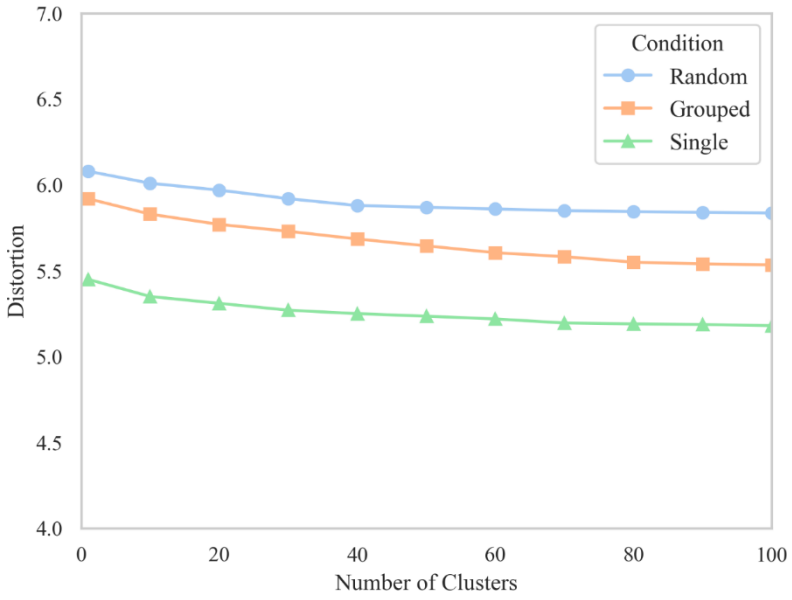

Fig 1.4 Cluster analysis on representations formed across 1000 different FEBAMs.

Therefore, different FEBAMs generated different representations when presented with the same pattern(s). Fig. 1.5 illustrates an example of this process, where *f* is the number of FEBAMs.

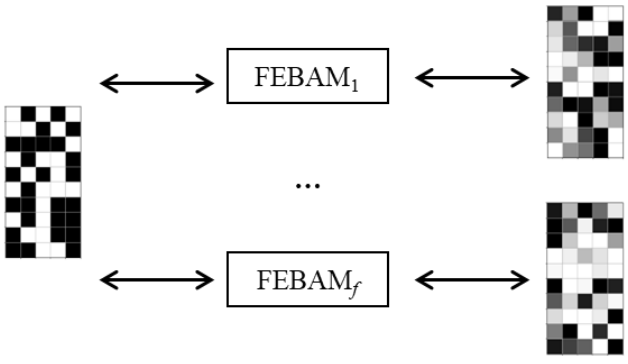

Fig 1.5 Generating representations for the "single" condition with different FEBAMs, where *f* is the number of different FEBAMs.

1.5 Simulation II: Exemplar categorization

In this section, we further investigate whether different FEBAMs respect the same behaviour during exemplar categorization. To do this, we extended the condition 2 of simulation I to five categories. However, in this case, clustering analysis will be conducted on individual FEBAMs and not all at once, as stated is Scenario B.

1.5.1 Methodology

The same method described in simulation I was used to generate input patterns. Here, two to five categories were generated. Each category contained five patterns. The correlation of patterns within the categories was approximately 0.95 and the correlation of patterns between the categories was set to approximately 0.15. The dimensionality of representations (u) was set again to 50. Each set of patterns were presented to 1000 different FEBAMs with the same learning procedure and parameters as previously described. Subsequently, following Scenario B, k-means cluster analysis was conducted on the generated patterns of individual FEBAMs only. Within-category and between-category correlation of generated representations were also examined. Finally, a recall test was performed to verify that patterns were correctly categorized.

1.5.2 Results

In Fig. 1.6, the mean number of clusters and distortion levels for the 1000 FEBAMs are presented. Results show that the generated representations respected the number of categories found in the input patterns (e.g. two categories, two clusters of generated representations). Furthermore, the average within-category correlation of generated representations was 0.75 and the average between-category correlation was <0.05. Lastly, the recall test yielded a performance of 100% correct pattern categorization.

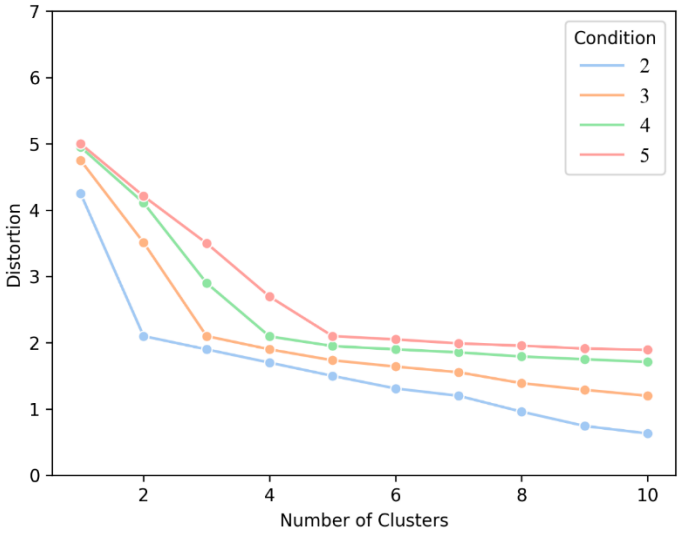

Fig 1.6 Average distortion and number of clusters for generated representations in function to the number of categories (condition).

In Fig. 1.7, a visual example this categorization is presented, where two sets of inputs are generated into two sets of representations where the within-category correlation is much larger than the between-category correlation. However, the within-category correlation between the representations is smaller than the within-category representations between the initial input.

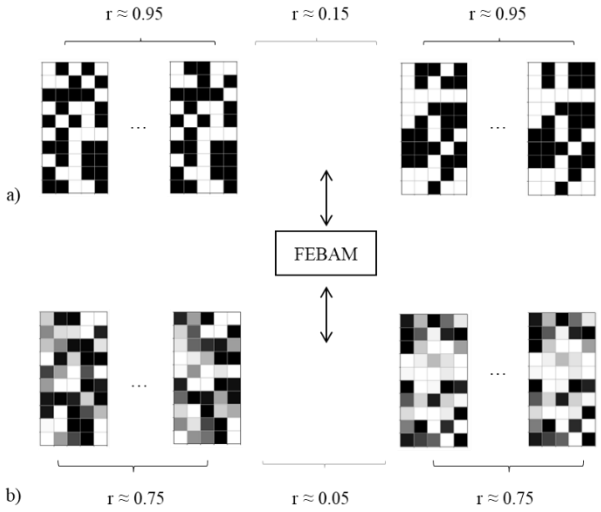

Fig 1.7 Example of exemplar categorization. Within category (black) and between category (gray) correlation for input (a) and output (b) patterns are presented.

1.6 Simulation III: Prototype categorization

In this last simulation, the goal was to examine the behaviour of the FEBAM during prototype categorization. A previous study showed that if the dimensionality of the representations is small enough when compared to the number of patterns, prototypes are formed (Giguère, Chartier, Proulx & Lina, 2007). However, the variability of recalled prototypes formed across different FEBAMs was not investigated.

1.6.1 Methodology

Two categories of input patterns, each containing five patterns, were generated in the same fashion as in Simulation Ib and II. The dimensionality of generated representations (*u*) was varied from 5, 10, 25, 50 to 100 dimensions. The patterns were presented to 1000 different FEBAMs using the same learning procedure and parameters as in simulation I and II. Two clustering analyses were conducted.

1.6.1.1. Simulation IIIa. First, to determine the relationship between distortion and size of representations, k-means cluster analysis was conducted on generated representations from individual FEBAMs. This was done following the procedure described in Scenario B.

1.6.1.2. Simulation IIIb. Second, to determine the variability of recalled patterns, k-means clustering analysis was conducted on generated representation of dimension 5 and their recalled patterns for all FEBAMs at once. This was done following the procedure described in Scenario A.

1.6.2 Results

Fig. 1.8 shows the first cluster analysis. The average number of clusters and distortion is presented. In all cases, two clusters were formed. Additionally, by lowering the *u*, clusters with lower

distortion began to appear. With representations of dimension 5, two clusters accounted for all the distortion, suggesting that prototypes were created.

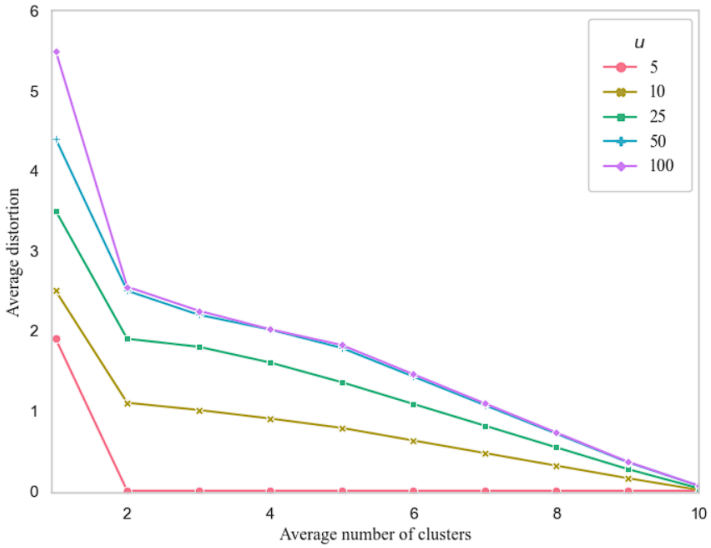

Fig 1.8 Simulation IIIa. Relationship between the number of clusters and number of y units.

Figure 1.9 presents the second k-means clustering analysis. Upon examining the representations produced by the 1000 FEBAMs, it becomes evident that there are no discernible clusters. This observation aligns with the findings from Simulation I, which suggest that different FEBAMs invariably produce distinct representations.

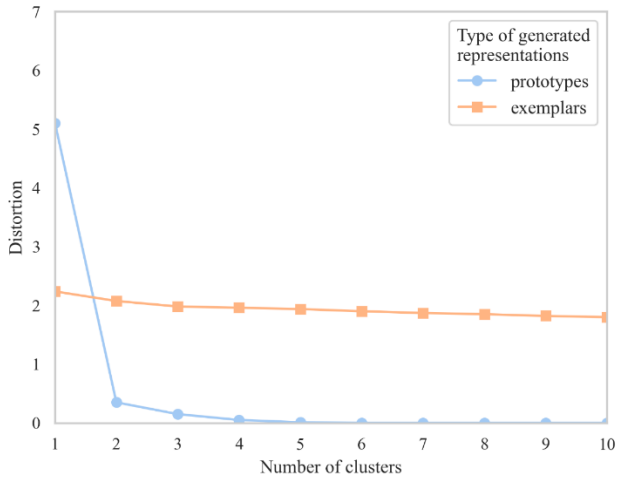

Fig 1.9 Simulation IIIb. Recalled prototypes and generated representations clustering across 1000 FEBAMs.

Additionally, an analysis of the recalled patterns reveals the presence of two clusters. Notably, these clusters are responsible for the vast majority of the observed distortion. This suggests that although coming from 1000 different FEBAMs, the same two patterns were recalled. Figure 1.10 illustrates an example of this process.

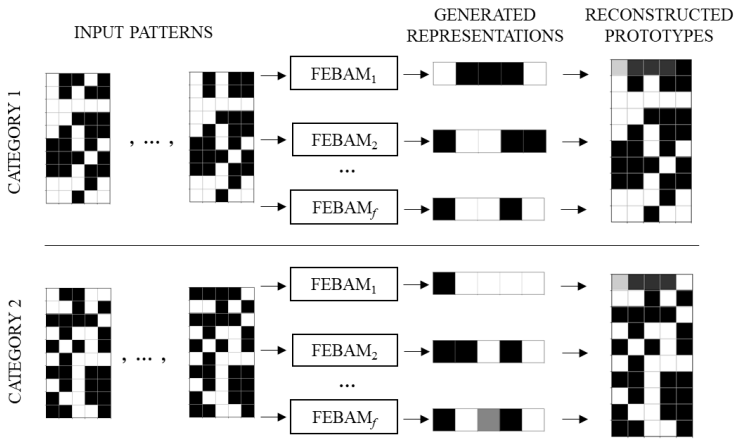

Fig 1.10 Example of Prototype categorization. Two categories of input patterns are presented to f different FEBAMs. These generated representations of dimension 5 (u = 5). Although representations are always different, the same two patterns are recalled. These recalled patterns act as a prototype for each input category.

1.7 Chapter I Discussion

This chapter aimed to determine if the FEBAM could shed light on the categorization process found within and between individuals. Results from Simulation I showed that when learning the same stimuli, different FEBAMs will generate diverse representations of these input patterns. As seen by the absence of clusters during a k-means clustering analysis. This result was expected since connection weights were initialized randomly. However, Simulation II found that although different FEBAMs generate different representations, their learning behaviour remained the same. The correlation of generated representations from each FEBAM first showed this. The within-category correlation (≈ 0.75) was far greater than the between-category correlation (≈ 0.05). This was further shown with a k-means clustering analysis on the representations. The analysis put forward that the number of clusters corresponded to the number of categories. In addition, recalled patterns were correctly categorized into individual exemplars. These findings propose a key feature that the FEBAM will have the same encoding behaviour even if the initial connection weights differ. This also contributes to previous work by showing that representations and reconstructed patterns are categorized similarly (Chartier et al., 2012; Giguère et al., 2007). This characteristic was further explored with prototype categorization.

Simulation III showed that different FEBAMs constructed the same prototypes even if the stored representations differed. If the size of the representations is equal to or lower than the number of patterns of a given category, then the same pattern is always recalled. This recalled pattern was a prototype of all input patterns within a given category. Thus, the network still created the same prototypes even if the initial learning conditions and subsequent generated representations were different. These findings are a promising step towards better understanding

how individuals exhibit common cognitive functionality despite variability in neural activity and may help define the optimal conditions to perform a classification task.

Future work could focus on how manipulating weight initialization may influence learning. A change in initial weight connections between different FEBAMs could result in a corresponding change in their generated representations. Furthermore, depending on the size of the network, the FEBAM exhibits different behaviours during the reconstruction of the input patterns (prototype or exemplar recall). An interesting property would be to grow (increase y-units) or prune (decrease y-units) the network based on a task. This would help to surpass the current task-specific problem and allow the model to be more generalized.

1.8 Chapter I concluding remarks

To sum up, this chapter showed that the FEBAM provides consistent categorization behaviour within the same architecture size, in either exemplar and prototype encoding, regardless of initial weight or network conditions. This finding was important, as it paves the way for multiple FEBAMs to be used to solve a task, without the need to manually specify the initial weights or modify the task inputs in a trial an error fashion. Therefore, since it can be concluded that only the architecture of the FEBAM changes its behaviour, the next chapter examined creating a multilayered network using multiple FEBAMs. As such, the generated representations from one FEBAM were used as input to another. The goal was to examine how the correlation between the generated representations would change as more and more FEBAMs are used.

2. Chapter II

Generating Cognitive Contexts with Feature-Extracting Bidirectional Associative Memory

Abstract

Associative learning is a key concept in human cognition that is fairly understood. However, when faced with similar patterns, challenges arise. In contrast, the mechanisms to deal with such challenges are not yet understood. In the field of artificial neural networks, the proposed cognitive contexts (CC) shed some light on the subject by using orthogonal contextual information to distinguish similar patterns. Although this method greatly improved learning and recall performances of similar patterns, it relied on the use of arbitrary components, limiting its cognitive plausibility. In this paper, a two-part experiment was performed to show that CC can be generated from highly correlated patterns by using a feature-extracting bidirectional associative memory (FEBAM). Results from the first experiment showed that the CC generated by the FEBAM can be retrieved under noise as effectively as orthogonal CC. On the other hand, one FEBAM cannot decorrelates the input patterns completely. As consequence, the second experiment consisted of using a network comprised of multiple FEBAMs to further decorrelate the generated CC. In this way, CC with a low correlation were achieved. CC of this correlation value would allow the same improvements in learning and recall performance found with orthogonal components. These findings confirm the usefulness of the FEBAM to generate cognitive contexts and are a promising path towards better understanding the cognitive mechanism behind learning highly similar stimuli.

Keywords: *Bidirectional associative memory, Feature extraction, Learning, Context, Correlated stimuli*

2.1 Introduction

The concept of learning and memory has always sparked much interest in the field of cognition. It is well known that to better understand its functioning, it is primordial to grasp the concept of associative memory (Nikouei Mahani et al., 2016). Associative memory consists of learning and storing pairs of identical (auto-association) or different (hetero-association) patterns. The strength of this approach resides in their inherent ability to reconstruct original patterns only from partial cues, a process akin to categorization and classification. Subsequently, the presentation of a new pattern will generate the retrieval of the most resembling stored pattern. However, when highly similar patterns must be stored separately, we are faced with two challenges. There is an increase in time to learn them and they are more prone to error during their recall. Nevertheless, given time, humans are capable of learning very similar patterns.

In cognition, the essence of associative memory has already been captured by various artificial neural networks (ANNs), as seen with models such as the bidirectional associative memory, or BAM (Chartier & Boukadoum, 2006a; Kosko, 1988). Nowadays, these can perform auto-association and hetero-association with accurate performances. Yet, models also face a challenge when learning similar patterns; highly correlated patterns. It was observed that the higher the correlation between the patterns, the poorer the performances will be during both learning and recall (Werker et al., 2002; Yoshida et al., 2009). In consequence, Rolon-Mérette et al. (2018) proposed a new technique which involved creating orthogonal cognitive contexts (CC) and juxtaposing them to highly correlated patterns to distinguish them. This resulted in the reduction of correlation between patterns without losing their original representation. This method decreased learning times while considerably increasing noise tolerance during recall. Moreover, it was shown

that there is no need for perfect orthogonal patterns to achieve high performance. A correlation lower than 0.15 is enough.

Although the use of orthogonal CC proved viable, the question remains: How are those CCs obtained? Thus far, they were provided from an external source and severely limited the model's internal consistency. One solution could be using the network to generate the right CC for encoding. The key to generating cognitive contexts following all these requirements may reside in unsupervised feature extraction techniques. Feature extraction models, such as neural PCA, are of interest since they can generate lowly correlated representations of input patterns (Fiori, 2000). From it, a recurrent associative memory model could link these representations to their corresponding input patterns. However, to have a good general cognition model, feature extraction (CC) and learning the augmented pattern (original and CC) must be accomplished under the same framework. In other words, both learning and transmission functions should be the same, while the architecture could differ. Furthermore, many of these feature extraction techniques change the representation of the input patterns from a bipolar to a grey-level representation, which makes the subsequent learning difficult if no arbitrary recording is allowed. Consequently, it is also important to use an ANN capable of dealing with grey-level patterns.

Currently, very few associative models can learn grey-level patterns without recoding the stimuli (Acevedo-Mosqueda et al., 2013). Chartier and Boukadoum (2006a) proposed a model to learn grey-level input without prior transformation. Moreover, this model was modified to perform feature extraction (Chartier et al., 2007). This feature extraction bidirectional associative memory (FEBAM) combines the ability of features extraction while also conserving the same learning and transmission functions like the BAM. Therefore, the use of both FEBAM and BAM could allow the creation of a combined network which could generate its own CC and use it to store the desired

association. On that account, the goal of this paper is to show the effectiveness of learning highly correlated patterns without losing the original representation with the use of generated cognitive contexts. This paper will be divided in the following fashion. In section 2 a brief background of the proposed model will be presented. Section 3 presents simulation I, where a FEBAM is used to generate cognitive contexts and assess its decorrelation level. Section 4 contains simulation II, which show how multiple FEBAMs can generate the right representation with the desired correlation. Finally, in section 5, the paper ends with a short discussion.

2.1 Model

The model used is the FEBAM, a modified BAM capable of generating representations from the inputs only and able to reconstruct them from those representations alone, even under various levels of noise. The internal mechanism of the FEBAM is like the one in BAM; they both have the same transmission and learning functions. The difference lies in the architecture. Contrary to the BAM, in the FEBAM there is no explicit teacher, meaning there is only one set of external connections, $\mathbf{x}_{[0]}$. The model is completely unsupervised; only one set of inputs is available. Like any other ANN, the network can be entirely described by its architecture, transmission and learning functions.

2.2.1 Architecture

The FEBAM architecture is illustrated in Fig. 2.1. The model has two layers of interconnected units in a bidirectional fashion, where the W and V layers return information to each other. Contrary to traditional BAMs, there is only one explicit connection, $\mathbf{x}_{[0]}$, to allow the network to perform feature extraction.

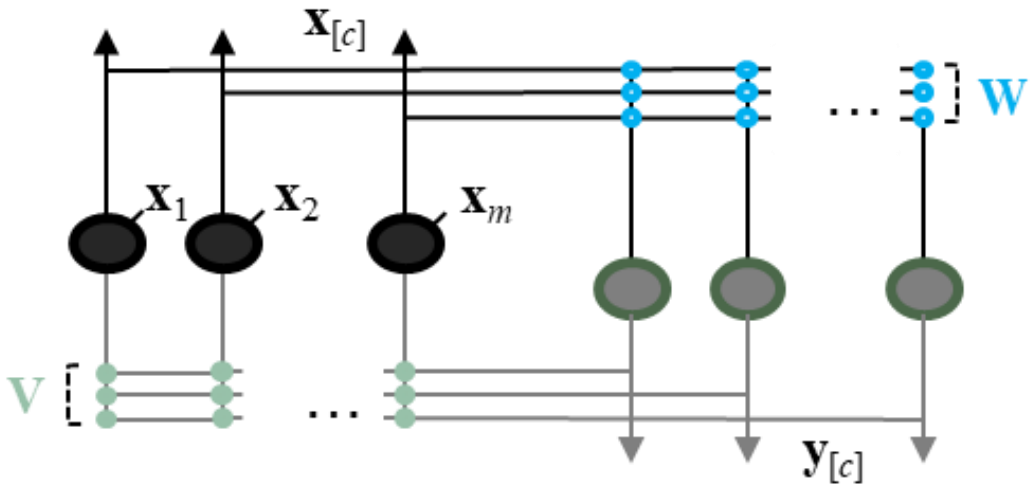

Fig. 2.1. Architecture of the FEBAM

2.2.2 Output function

The output function, also known as a transmission function, used is a cubic polynomial with bipolar limits at -1 and 1, defined by equations 2.1 and 2.2):

$$(2.1)\ \forall i, \dots, n, \mathbf{y}_{i[c+1]} = f(\mathbf{a}_{i[c]}) = \begin{cases} 1, if\ \mathbf{a}_{i[c]} > 1 \\ -1, if\ \mathbf{a}_{i[c]} < -1 \\ (\delta + 1)\mathbf{a}_{i[c]} - \delta \mathbf{a}_{i[c]}^3, Else \end{cases}$$

$$(2.2)\ \forall i, \dots, m, \mathbf{x}_{i[c+1]} = f(\mathbf{b}_{i[c]}) = \begin{cases} 1, if\ \mathbf{b}_{i[c]} > 1 \\ -1, if\ \mathbf{b}_{i[c]} < -1 \\ (\delta + 1)\mathbf{b}_{i[c]} - \delta \mathbf{b}_{i[c]}^3, Else \end{cases}$$

Where n and m are the total number of units in each layer, i is the index unit, δ is the general transmission parameter, c is the iteration cycles index and **a** and **b** are the activations, obtained with $\mathbf{a}_{[c]}=\mathbf{W}\mathbf{x}_{[c]}$ and $\mathbf{b}_{[c]}=\mathbf{V}\mathbf{y}_{[c]}$. Figure. 2.2 shows the transmission function for δ= 0.2.

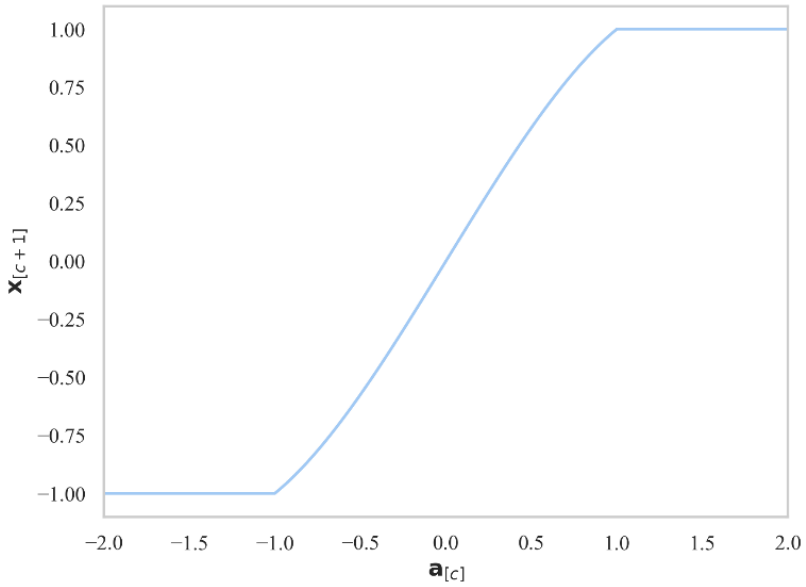

Fig. 2.2. Transmission function for δ= 0.2.

2.2.3 Learning function

The connection weights are modified following a hebbian/anti-hebbian rule (Storkey & Valabregue, 1999) following equations 2.3 and 2.4:

$$(2.3)\ \mathbf{W}_{[k+1]} = \mathbf{W}_{[k]} + \eta\left(\mathbf{y}_{[0]} - \mathbf{y}_{[c]}\right)\left(\mathbf{x}_{[0]} + \mathbf{x}_{[c]}\right)^{\mathrm{T}}$$

$$(2.4)\ \mathbf{V}_{[k+1]} = \mathbf{V}_{[k]} + \eta\left(\mathbf{x}_{[0]} - \mathbf{x}_{[c]}\right)\left(\mathbf{y}_{[0]} + \mathbf{y}_{[c]}\right)^{\mathrm{T}}$$

In this context, $\mathbf{x}_{[0]}$ represents the initial inputs and $\mathbf{y}_{[0]}$ signifies the initially set of generated cognitive contexts (CC). $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$ are their corresponding reconstructions after undergoing c cycles. The learning parameter is denoted by η, and k represents a specific learning trial. According to Equations (2.3) and (2.4), convergence of the matrix weights is achieved only when $\mathbf{x}_{[0]}$ equals $\mathbf{x}_{[c]}$ or $\mathbf{y}_{[0]}$ equals $\mathbf{y}_{[c]}$. To optimize simulation efficiency, the number of iterations is typically limited to c = 1. Based on previous research by Chartier and Boukadoum (2006a), it is established that learning will converge provided the learning parameter (η) is set following:

$$(2.5)\ \eta < \frac{1}{2(1-2\delta)\mathrm{Max}[m,n]}, \delta \neq \frac{1}{2}$$

During learning, $\mathbf{y}_{[0]}$ are not initially available. Instead, they are obtained after a first iteration through the network, hence they are generated representations that can be used as CCs. Figure 2.3 shows this process, where $\mathbf{y}_{[0]}$ is obtained by the iteration of $\mathbf{y}_{[0]}$ through the corresponding weight connections W using the transmission. Similarly, $\mathbf{x}_{[1]}$ will be obtained from $\mathbf{y}_{[0]}$ and $\mathbf{y}_{[1]}$ from $\mathbf{x}_{[1]}$. Through the weight updates, each $\mathbf{x}_{[c]}$ will converge to a solution that will try to best reconstruct its associated initial input $\mathbf{x}_{[0]}$. The number of units in the $\mathbf{y}$–layer determined the dimensionality of the generated representations. Normally, the more units there are, the better the reconstruction will be (Giguère et al., 2007; T. Rolon-Mérette et al., 2019).

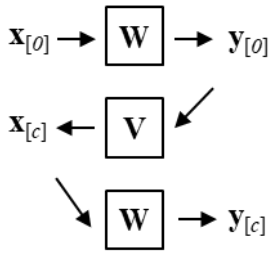

Fig. 2.3. An iterative cycle during learning to obtain $\mathbf{y}_{[0]}$, $\mathbf{x}_{[1]}$ and $\mathbf{y}_{[1]}$ from the initial inputs $\mathbf{x}_{[0]}$ for weight updates.

2.3 Simulation I – Generating Cognitive Contexts

First, a FEBAM was used to generate cognitive contexts from highly correlated patterns. The objective was to assess the level of decorrelation that a single FEBAM could achieve. Second, performance evaluations were made between the FEBAM and BAM under noise. This comparison allowed to determine if there is a difference between generated CC and arbitrary orthogonal CC.

2.3.1 Methodology

Two sets of 16 correlated vector inputs were produced by generating bipolar-valued vectors. These inputs can be represented as pixelated patterns where black pixels represent the value of +1 and

white pixels -1. The dimensionality of inputs was kept constant at 64 and the correlation between patterns was either 0.94 (high) or 0.50 (medium).

2.3.1.1 Learning

The correlated bipolar patterns were presented to a FEBAM. The transmission function parameter (δ) was set to 0.2 and the learning parameter (η) was set to respect equation (4). Weights were randomly initialized with values between -0.1 and 0.1. To assess the effect of convergence of the generated features two conditions were studied. Whether an initial pattern has to cycle once (c = 1) before the weights are updated (Figure 2.3) or five times (c = 5). A higher number of iterations were also studied but since the results do not significantly differ from c = 5 they were omitted. Learning stopped when the network achieved a mean squared error (MSE) of less than 10-9 and was conducted following this procedure:

1. Creation of a random list of patterns with the correlation and dimension previously mentioned.
2. Random selection of a given pattern from the list to obtain $\mathbf{x}_{[0]}$.
3. Iteration through the network (as illustrated in Figure 2.3) according to c cycles using the output function (1a and 1b) to obtain $\mathbf{y}_{[0]}$, $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$.
4. Computation of weight updates according to the learning rule (2.3 and 2.4).
5. Repetition of steps 2) to 4) until the minimum mean squared error between $\mathbf{y}_{[0]}$ and $\mathbf{y}_{[c]}$ was reached.

2.3.1.2 Recall

Recall performances were obtained using a pixel flip noisy recall task according to the following procedure:

1. Selection of an initial input from list $\mathbf{x}_{[0]}$.
2. Perform random pixel flips to add noise to the input selected.
3. Compute x(t) in accordance with the transmission function (2.1 and 2.2) until convergence.
4. Comparison of this output with the correct one and calculate the performance.
5. Repetition of step 1) to 4) for 200 trials.
6. Repetition of step 1) to 5) for each initial stimulus.
7. Repetition of step 1) to 6) with increasing ratio (0 to 50%) of random pixel flips.

Under the same recall task, results were compared to performances of orthogonal CC associated to the same highly correlated patterns in a BAM network. A modified BAM was used called the Bidirectional Hetero-associative Memory (BHM) as the learning procedure for the BHM is very similar to the FEBAM's and they both share the same internal functions. Details for the BHM can be found in detail jn (Chartier et al., 2007). Fig. 2.4 shows the hetero-association between inputs and CC for both models.

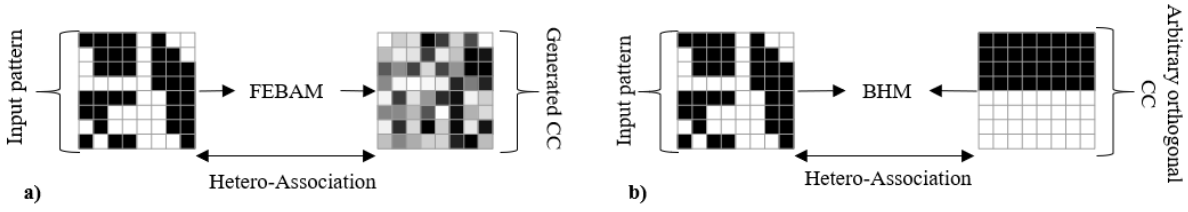

Fig. 2.4. Hetero-association process via a) FEBAM and b) BHM

2.3.2 Results

Results are shown in (Fig. 2.5). From inputs with an initial correlation of 0.94, the FEBAM generated cognitive contexts with an average correlation of 0.75. Increasing the iteration from one to five resulted in CC of correlation 0.50. For inputs of medium correlation (0.50), the FEBAM generated CC with an average correlation of 0.15. Increasing the iteration from one to five resulted in cognitive contexts of correlation 0.12. Thus, increasing the number of cycles before each weight update allowed the generated CC to have a lower correlation. Nevertheless, a single FEBAM was not able to decorrelate the highly correlated inputs. In the case of noisy recall where random pixels were flipped, the performance of both the FEBAM and the BHM decreased as the level of noise increased. Not surprisingly, recall performances were better for patterns of moderate correlation (0.50) than high correlation (0.94). Both BHM and FEBAM exhibited very similar performances even though the associated patterns were different. Overall, the FEBAM can generate CC from highly correlated input patterns. The fact that a single FEBAM couldn't decorrelate the inputs had no impact on the performance. Hence, confirming that the recall performance is solely a function of the correlation of the input patterns.

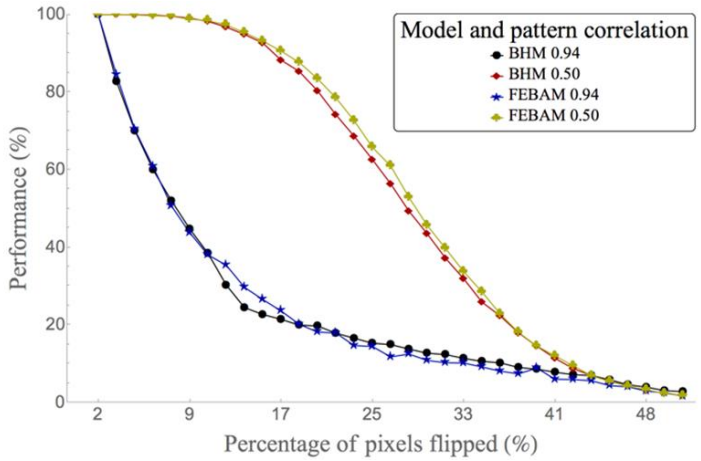

Fig. 2.5. Performance level of the pixel flip recall task. Results for FEBAM and BHM hetero association between patterns and CC are presented with patterns of 0.94 and 0.50 correlation.

2.4 Simulation II – Multiple FEBAMs

As shown in simulation I, it is possible to generate CC with a correlation of around 0.75 from highly correlated patterns. Also, increasing the iteration to five resulted in CC with correlation 0.50. However, these correlation levels are not sufficient to achieve the desired level of 0.15 . In this simulation, the goal was to lower the correlation between CC with the use of multiple FEBAMs, one after the other. In this way, they will be more effective to decrease learning times and increase recall performances of highly correlated patterns during an associative task. Furthermore, to have a better understanding of the process, the number of iterations, dimensionality and memory load were also investigated.

2.4.1 Methodology

Pixel-based patterns were created in the same fashion as simulation I. However, these bipolar patterns were varied at different levels of correlation, size or memory load. Correlation was varied at either 0.94, 0.73 or 0.50 (Fig. 2.6) to represent high to medium levels of correlation of input patterns. Additionally, three different numbers of patterns (n), with dimensionality (d) kept at 64, were used to test the effects of memory load (n/d). More precisely, memory loads of 25%, 50% and 75% were used. Finally, the highly correlated patterns (0.94) varied in dimensions of 64, 128 or 256, while the memory load was kept constant at 25%.

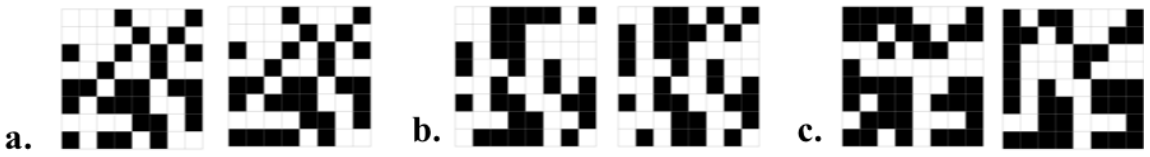

Fig. 2.6. Example of two input patterns representing correlations of 0.94(a), 0.73(b) or 0.50(c).

To generate cognitive contexts with low correlation, multiple FEBAM networks were used. An initial set of patterns was presented to the first FEBAM, generating cognitive contexts $CC_{(1)}$. These contexts were fed to a second FEBAM, generating a second set of contexts $CC_{(2)}$. This process was repeated until (f =10) FEBAMs were used (Fig. 2.7). In each step, correlation between generated CC was recorded. Learning was conducted in the same fashion as simulation I, with the same choice of free parameters.

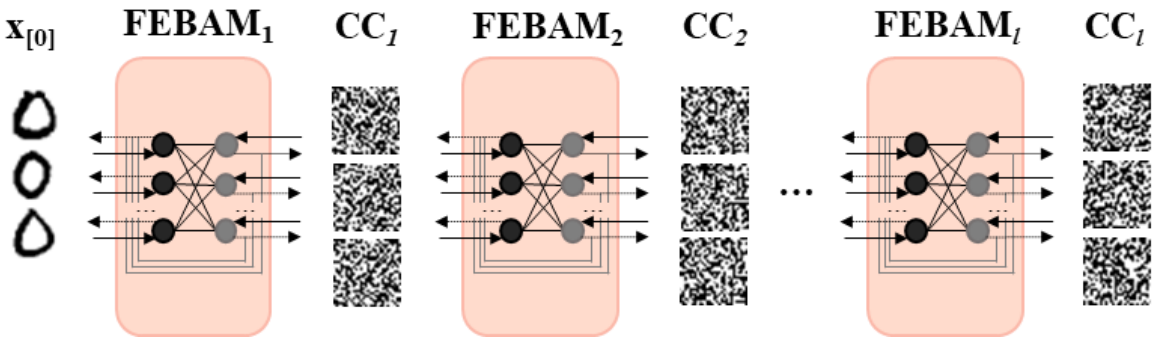

Fig. 2.7. Multiple FEBAM process.

2.4.2 Results

Fig. 2.8 shows the average correlation of CC generated by multiple FEBAMs. We can see that by increasing the number of FEBAMs, it is possible to lower the correlation of the generated CC even further than the 0.75 originally obtained when using one FEBAM. In fact, the correlation approaches asymptotically to 0 as more FEBAMs are used.

This asymptotic characteristic varied slightly depending on the modalities of initial patterns. Unsurprisingly, as the correlation of initial patterns decreased, decorrelation increased (Fig. 2.8a). On average, for a high correlation (0.94) between inputs patterns, the generated CC reached a correlation of 0.15 after four FEBAMs compared to two FEBAMs for moderate correlation (0.50). Second, the impact of memory load on the decorrelation of CC was not significant (Fig. 2.8b). In contrast, Fig. 2.8c shows that as the size (dimension) of stimuli increased,

the correlation of generated CC decreased at a slightly quicker rate. On average, for highly correlated patterns with a dimension of 256, a correlation of 0.15 between CC was achieved after three FEBAMs compared to four FEBAMs with lower dimensionality (64). Finally, increasing the iteration cycle to five also accelerated decorrelation for highly correlated patterns as 0.15 was reached after three FEBAMs (Fig. 2.8d).

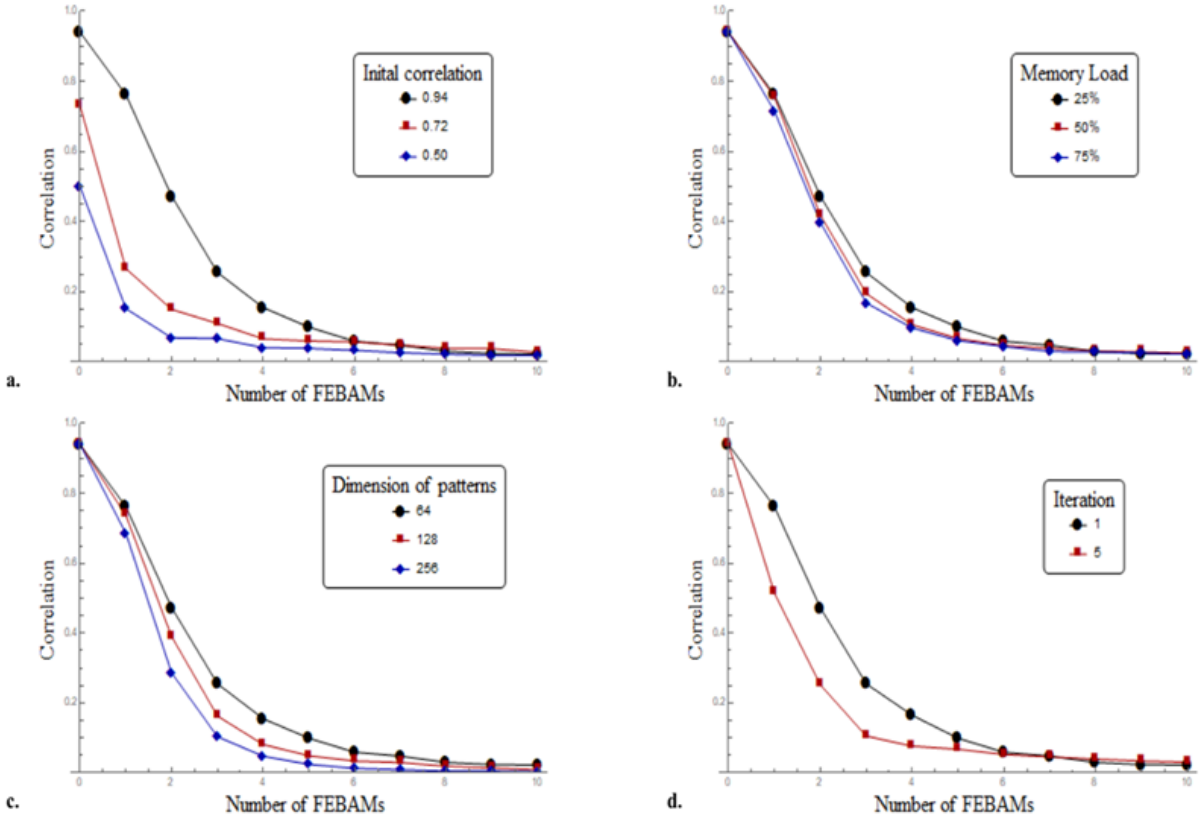

Fig. 2.8. Correlation of generated cognitive contexts in function of the number of FEBAMs for different initial patterns (a-c) or iteration cycle (d). Results shown for: (a) different correlation of initial patterns, (b) different memory load, (c) varying size of patterns and (d) iteration cycles of one or five.

2.5 Chapter II Discussion

During Simulation I, the FEBAM generated CC from highly correlated patterns. It was shown that the model correctly associated each generated CC to its corresponding input patterns. It also exhibited very similar performances to the BHM during a noisy recall task. This is promising since there is a clear advantage of using the FEBAM to generate CC over using the BHM and arbitrary CC, as previously stated. These generated CC were found to have lower correlation than the input patterns going on average, from 0.94 to 0.75. Furthermore, it was shown that by increasing the number of iterations to five, the correlation of the generated CC decreased up to 0.50.

Nonetheless, these correlation values were not sufficient. As mentioned, the correlation between CC had to be lower than 0.15 to achieve optimal learning and recall performances (Rolon-Mérette et al., 2018). In section 2.4, it was shown that this correlation could be lowered even further by using a network comprised of multiple FEBAMs. The relationship between correlation of generated CC and number of FEBAMs was asymptotic as the correlation approached 0. This decorrelation was observed with various sets of inputs patterns varying either in correlation, memory load or dimensionality. Among these, dimensionality had an interesting effect, the higher the dimensionality the higher the decorrelation of CC. This property could lead to the production of the desired value of decorrelation for a given situation. It could even be argued that this process is similar to the level of expertise, where the ability to discriminate similar patterns is better with higher expertise and resources (Tervaniemi et al., 2009). In addition, the decrease in correlation previously observed with an iteration of five continued when applied to multiple FEBAMs. The number of iterations and dimensionality of patterns are interesting facets to focus on for future studies. Once the CC are generated, they can be juxtaposed to highly correlated patterns prior to learning. Subsequently, by increasing the relative size of generated CC, it is possible to reach a

correlation below 0.15 after juxtaposition to highly correlated patterns. If CC of correlation 0.10 are used, their dimensionality would only need to be two to four times bigger than the orthogonal CC to achieve similar results. Because of this and the fact that the FEBAM presents the same core properties as the BHM, we get an overall model with a good internal consistency.

Although it was shown that the network can generate its own CC, there is currently no cue to initiate such process. If the goal was to correctly model the cognitive mechanisms behind learning highly correlated stimuli, it is important that future models juxtapose generated cognitive contexts on their own. By doing so, difficult tasks, such as the one-to-many associations, could be completed. On the flip side, a low correlation of generated CC might not be desired depending on the task at hand. If the goal is to regroup dissimilar patterns, then the use of highly correlated cognitive contexts could aid. This could be achieved by lowering the number of y output units in the FEBAM, thus increasing the CC correlation. This process could be useful for learning in a noisy environment where small perturbation should not create a new item in memory as it is currently the case in BAMs. Finally, it is important to note that the CCs generated from highly correlated input patterns did not modify the original inputs which contrast with other methods, such as neural PCA. Additionally, the bidirectional nature remains even in a multi-layer setting, meaning the original inputs can be recalled. Such advancements also offer deeper insights into the cognitive processes underpinning learning and memory.

2.6 Chapter II concluding remarks

The capability of the FEBAM to extract features positions it as an excellent foundation for a multilayered network. This network self-generated representations that are significantly less correlated, making it an artificial neural network that can adeptly handle highly correlated stimuli. Within this chapter, these self-generated representations produced by various layers were analyzed

purely by their correlation. Nonetheless, it is conceivable that these representations undergo a progressive segregation into a higher-dimensional space, transforming inputs from non-linear tasks into linearly separable formats. Pursuing this hypothesis, the subsequent chapter combined the multilayered FEBAM structure with a Binary Associative Memory (BAM), the supervised version of the FEBAM that can only learn linear separable associations. As such, this fusion meant that the BAM does not rely on raw inputs from non-linear tasks but rather on the generated representations from the multilayer FEBAM. The emergent model was a multilayered Associative Memory, competent in mastering non-linear tasks while maintaining consistent internal dynamics throughout its structure, thereby preserving the cognitive attributes associated with local and Hebbian learning found in individual BAMs and FEBAMs. This was examined with multiple non-linear tasks and evaluation metrics.

3. Chapter III

A Multilayered Bidirectional Associative Memory Model for Learning Nonlinear Tasks

Abstract

A multilayered bidirectional associative memory neural network is proposed to account for learning nonlinear types of association. The model (denoted as the MF-BAM) is composed of two modules, the Multi-Feature extracting bidirectional associative memory (MF), which contains various unsupervised network layers, and a modified Bidirectional Associative Memory (BAM), which consists of a single supervised network layer. The MF generates successive feature patterns from the original inputs. These patterns change the relationship between the inputs and targets in a way that the BAM can learn. The model was tested on different nonlinear tasks, such as the *N*-bit, Double Moon and its variants, and the 3-class spiral task. Behaviors were reported through learning errors, decision zones, and recall performances. Results showed that it was possible to learn all tasks consistently. By manipulating the number of units per layer and the number of unsupervised network layers in the MF, it was possible to change the level of nonlinearity observed in the decision boundaries. Furthermore, results indicated that different behaviours were achieved from the same set of inputs by using the different generated patterns. These findings are significant as they showed how a BAM-inspired model could solve nonlinear tasks in a more cognitively plausible fashion.

Keywords: *Artificial neural Networks, Associative Memory, Cognition, Multilayer, Nonlinear Tasks*

3.1. Introduction

According to the associative memory theory, the human brain is an association machine (Anderson & Bower, 2014; Buckner & DiNicola, 2019). Every day, it is learning, storing, and retrieving associations between objects, concepts, and other mental items that can be vastly different (Caudill & Butler, 1990). This process, referred to as Associative Memory (AM), is said to be imprinted in various cognitive behaviours and has been studied extensively in a wide array of fields (Kumar et al., 2015; Lansner, 2009; Letzkus et al., 2015; Leuner et al., 2003; Mayes et al., 2007; Puig et al., 2014; Stalnaker et al., 2010; Suzuki, 2008). Of particular interest is the computational modelling domain, which has grown in popularity over the last decades. Specifically, Artificial Neural Networks (ANNs) have been widely used to draw parallels between neurophysiology and cognition in many different settings (Elman, 1996; Smolensky, 1988; Yang & Wang, 2020). One of their many benefits is that they permit one to infer how neural information processing can lead to certain cognitive properties, like AM (Alamia et al., 2020; Barak, 2017; Hintzman, 2014; McClelland, 2000; O'reilly & Munakata, 2000; Thomas & McClelland, 2008).

3.1.1. Background and Problematic

ANNs built for studying AM are often referred to as Neural Associative Memory (NAM) (Palm, 2013; Zhang et al., 2021). Like ANNs, most NAM also consist of neuron-like and synapse-like elements based on the McCulloch-Pitts neuron (McCulloch & Pitts, 1943). NAMs play an important role in cognition. They provide an interesting explanation for how learned patterns may correspond to attractors in the brain’s neuronal state space. As such, they permit linking the neurodynamic perspective to AM and consequently find themselves in various brain theories (Knoblauch, 2005). Most NAMs use content addressable memory, a type of computer memory

that searches for pattern similarity among its stored content through a process akin to AM. This allows them to retrieve a corresponding address pattern despite being given a noisy one, a feature incorporated in most memory models in cognition (Kohonen et al., 1977). NAMs can usually perform either auto-associations, hetero-associations, or both, which makes them interesting when trying to explain various cognitive behaviours through AM, see the HAM model by Anderson and Bower (2014) for details on how associations can be used to build cognition. As such, one of their many benefits is that they permit to explain how information signal processing can lead to human cognition.

Remarkably, NAMs are nothing new and can be traced back to the 60s, albeit limited and far from displaying proper AM-like behaviours (Steinbuch, 1961; Willshaw et al., 1969). These behaviours only began to emerge in the 70s, with networks being able to perform auto- and hetero-associations (Kohonen et al., 1977; Palm, 1980). However, these models lacked the ability to form attractors due to their internal dynamics. This came with the brain-state-in-box, or BSB (Anderson et al., 1977). The BSB could perform auto-association by storing patterns in attractors that were produced by nonlinear recurrent feedback within its network. Consequently, the BSB inspired a new subclass of NAMs, referred to as recurrent neural associative memory (RNAM), mainly interested in the neurodynamic perspective and cognition (Anderson, 1983).

RNAMs became popularized a few years later due to the Hopfield network (Hopfield, 1982). Although this model showed how correlated stimuli could be learned through a simple one-shot learning rule and proved this stabilization, the model could only display auto-association. A few years later, this limitation was overcome with the Bidirectional Associative Memory (BAM), which could perform both auto-association and hetero-association while retaining its neurodynamic perspective (Kosko, 1988).

To this day, the BAM remains one of the most popular RNAM for studying AM through the neurodynamic perspective. This is because it can perform auto- and hetero-association in a bidirectional fashion while using a simple Hebbian learning rule, essential aspects for generalizing an ANN to human cognition. Over the years, numerous studies have enhanced the original BAM by allowing it to perform various complex tasks and be used for real-world applications, such as data compression, image recognition, and classification (Acevedo-Mosqueda et al., 2013). As with ANNs in general, these advancements mainly came from two perspectives: those interested in performance exclusively, often called the Machine Learning (ML) approach, and those searching to build more plausible models of mechanisms found in the animal brain, occasionally denoted as the cognitive or biological approach. Although the ML perspective allowed the BAM to be used in the aforementioned applications, it did so at the cost of losing some of its core properties related to cognitive plausibility: biological realism, distributed representations, error-driven learning, Hebbian learning, and bidirectional activation propagation (O'Reilly, 1998).

On the flip side, cognitively plausible models that try to follow these principles are often seen as either too complex (Labib, 1999), requiring highly nonlinear transmission functions (Shen et al., 2004), using subjective learning procedures (O'Reilly, 1996) or in the BAMs' case, limited when it comes to learning nonlinear associations (Chartier et al., 2009). This is quite cumbersome for the generalizability of the BAM, especially when considering that humans can associate patterns regardless of the nature of the association being linear or nonlinear (Levering et al., 2020; Rosedahl & Ashby, 2022). For example, when collecting wild mushrooms, each mushroom encountered will have characteristics that allow an individual to determine whether it is edible, like the colour, smell, and size (Baroni, 2017). In some instances, a single feature (foul smell) will be enough to determine that a mushroom is inedible, regardless of all other features. As shown in

Fig. 3.1a, this represents a linear association between the mushroom's feature and its edibility category. However, in most cases, many features must be considered simultaneously. Dealing with such situations often necessitates the ability to form nonlinear associations between features and their category, as shown in Fig. 3.1b. Although humans constantly do this, it remains an important challenge for cognitively plausible BAMs. Indeed, when these models try to learn a nonlinear task, such as the XOR problem, they remain incapable of finding the appropriate solution. A standard BAM's behaviour, illustrated in Fig. 3.1c, will be linear even when encountering a nonlinear task, meaning in the mushrooms example, the model will miss identify them. Suppose the BAM is to remain a model of human cognition through the neurodynamic perspective. It must be able to learn linear and nonlinear association types.

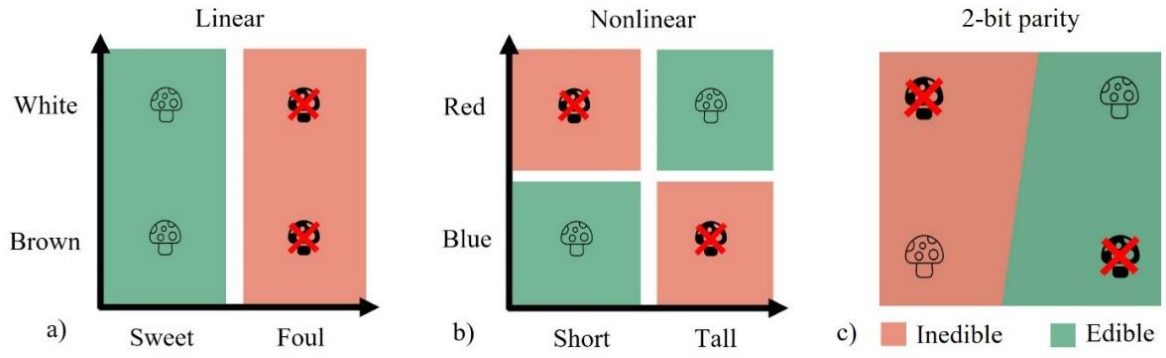

Fig. 3.1. (a) Example of linear associations. (b) Example of nonlinear associations. (c) A standard BAM's classification behaviour on the XOR task.

3. 1. 2. Related Work

Different efforts have gone towards teaching the BAM to learn nonlinear tasks. These usually consist of either modifying the BAM itself (e.g., changing its learning rules, architecture, transmission function, etc.) or pre-processing the inputs before being presented to the BAM (i.e., changing their representations). The most popular methods for dealing with nonlinear tasks consist of using multilayered architectures while adjusting learning through backpropagation and gradient

descent (Adigun & Kosko, 2019; Kosko, 2021; Liu et al., 2019; Rumelhart et al., 1986; Singh & Saraswat, 2017; Wu et al., 2018). The popularity of these methods can be attributed to the much-documented accomplishment of "deep" networks in numerous successful applications (Ferrucci et al., 2010; Silver et al., 2016; Vinyals et al., 2019). Deep networks are built upon the abstraction of the brain's multilayered architecture, which, in turn, could be an underlying reason for their immense power. However, these models are not necessarily concerned with following cognitive constraints and instead aim to solve tasks optimally (Marcus, 2018). The main problem with this approach is that it often requires different learning or transmission functions within the combined model while also having to freeze some of its modules during learning (Liu et al., 2019; Wu et al., 2018). On the other hand, when the BAM is expanded upon to include multiple layers within its architecture and uses a form of bidirectional backpropagation learning algorithm (Adigun & Kosko, 2019; Kosko, 2021; Singh & Saraswat, 2017), it assumes that information can circulate through the same connection bidirectionally. Moreover, its weight update usually requires an unrealistic flow of information, having to circulate the incoming signals many times between inputs and outputs layers before learning can occur.

Regardless of which approach above is used, they go against the current understanding of the human brain. For instance, at both the micro and macro level, networks of neurons seem to behave similarly throughout the brain (Buckner & DiNicola, 2019), meaning that the same transmission function should be used for all nodes, and internal consistency must be sought out. Moreover, there appears to be no evidence of brain regions stopping their activity from accommodating learning (Raichle, 2010). Thus, no weight-freezing schemes must be used. Additionally, information at the synapse level seems to only circulate in one direction, so no transposition of weight matrices for backward passes can be utilized (Südhof, 2021). Lastly, there

still needs to be more evidence that the strength of synapses changes based on the error being backpropagated through them (Song et al., 2020). Thus, although these "deep networks" approaches permit the BAM to learn nonlinear associations, they often do so in ways that undermine its plausibility and weaken its legitimacy to be used as a tool to model human cognition.

A different approach can be found in the morphological bidirectional associative memory, or MBAM (Ritter et al., 1999), gray-scale morphological associative memories (Sussner & Valle, 2006), the alpha-beta bidirectional associative memory (α-β BAM) (Acevedo-Mosqueda et al., 2006), chaotic complex-valued bidirectional associative memory (CCBAM) (Yano & Osana, 2009), Lie Algebra-Valued BAM (LieBAM) (Popa, 2017), and other variants. See (Acevedo-Mosqueda et al., 2013) for a chronological history of other relevant BAMs. These models have been built to deal with nonlinearity and have interesting properties, from the ability to increase storage capacity (Lee et al., 2015), display chaotic behaviours, to using drastically fewer resources (artificial nodes) in the process (Sussner & Campiotti, 2020). However, these models do so by changing the nature of the artificial node and using different algebra types for its computations. As such, they move away from the traditional view of the McCulloch-Pitts' neuron and the simple neurodynamic perspective captured by the original BAM.

An important question to be answered is whether changing the connectivity is sufficient to learn nonlinear tasks while keeping simple Hebbian learning and the McCulloch-Pitts' neuron model. One approach that investigated this can be found in (Chartier & Boukadoum, 2006a). This modified version removed the original notion in Koskos' BAM (1988), where the bidirectional connections between two units had to be symmetric. Instead, it used two distinct weight matrices to account for the independence of the forward and backward connections. In this modified BAM, a simple Hebbian and anti-Hebbian learning rule is used. From this model, an unsupervised

version was proposed by replacing one set of connections with its generated representation (Chartier et al., 2007). This variation, called the Feature Extraction Bidirectional Associative Memory (FEBAM), has been shown to learn nonlinear associations if combined with another FEBAM in a head-to-toe fashion (it solved the 2-bit, 3-bit, 4-bit, and 5-bit parity problem). This was evidence that learning these types of associations was possible by only changing the flow of information. In other words, success without undermining the model's internal consistency and while maintaining the modified BAM's core properties (Chartier et al., 2012). However, this method was only applicable if bipolar values represented the task and, as such, did not generalize well to continuous values. Nevertheless, the FEBAM can be considered an interesting model, as it has the property to generate patterns associated with the original inputs in an unsupervised manner but of different dimensionality to them. Thus, the FEBAM can generate a version of the initial patterns that are either compressed, expanded, or the same size (Giguère et al., 2007). This is similar to what can be observed in other ML models, such as autoencoders (Bank et al., 2023). In other words, this means that the FEBAM could be used as a type of pre-processing network.

This was explored in a study that combined the FEBAM and the BAM as a multinetwork model (Tremblay et al., 2013). Here, the FEBAM was used to generate a higher dimensionality representation that was joined with the original input to make the task linear. This FEBAM-BAM implementation proved interesting as it could learn the *N*-bit parity problem and other real-valued (continuous) nonlinear tasks (e.g., the double Moons). However, the downside of this multinetwork model was that its learning was erratic and required an external stopping mechanism to halt its weight update. Additionally, even with this stopping mechanism, the model could only find the right solution a fraction of the time. As such, although the FEBAM-BAM approach had internal consistency, it lacked reliability and the capacity to self-stabilize.

Still, this combination of FEBAM and BAM showed promise, as it pointed towards an advantage of using networks of the same nature. By allowing the multiple networks to work together and by only modifying the connectivity of one of them, it was possible to display new behaviours. Using such restrictions, the multimodel gets closer to being analogous to information circulating through different brain regions (neural circuitry) and being transformed along the way (Wang & Cui, 2018). As such, it may be an inherent explanation of how these homogenous systems can display more complex behaviours.

3. 1. 3. Proposed Solution

In a previous study looking at how to decorrelate highly similar stimuli, a model composed of several FEBAMs was used. Results showed that as the number of FEBAMs increased, the correlation between the generated representations decreased (Rolon-Merette et al., 2018). The human brain's laminar architecture inspired this multi-FEBAM model; as such, it was structured by using various versions of itself stacked one on top of each other. Its decorrelation properties functioned by generating an initial expanded representation of the inputs from the first FEBAM. This generated pattern was then given to the second FEBAM to generate a representation from it. This process was repeated until the last generated representation from the final FEBAM was obtained. Each subnetwork was trained sequentially in isolation, meaning the first FEBAM had to finish learning before the second one began. Although this multiple FEBAM was highly successful in decorrelating inputs, there may be an additional property that was not foreseen. More precisely, it may also be progressively transforming nonlinear relationships into linearly separable ones in a reliable fashion. This could be possible due to the FEBAM's ability to generate patterns through information expansion. Generating representation from projecting inputs onto a higher dimension could result in finding a linearly separable solution to a nonlinear problem, like what can be

observed in other ML models, such as the support vector machine (SVM) (Chauhan et al., 2019). By repeating this process over multiple FEBAMs, this generated representation can be "shuffled" until a set of patterns can share a linear association with its corresponding targets. Therefore, using this multi-model before the BAM as a pre-processing network could allow the latter to learn a task regardless of the level of nonlinearity. In other words, it would be possible to maintain the internal consistency displayed by the FEBAM-BAM approach while also showing reliability and self-stabilization.

Because the learning stabilizes from any starting condition, multiple FEBAMs could be linked together, such as each FEBAM would send its generated representations to the following one. By default, this sequential learning will require additional time because each FEBAM layer must receive constant information for learning to stabilize. At first, only the first network layer would receive constant information (the inputs) during the first few learning epochs. As this initial network layer generates the same representations, the following layer can then begin to produce a stable solution. This process would trickle down until the last FEBAM layer stabilizes. Hence, as a given layer begins to provide a steady solution, learning (weight update) can continue without hindering the model. In other words, by using a head-to-toe architecture and giving more time for each layer to provide a stable solution, it should be possible to exploit the feature-extracting abilities of the multiple FEBAMs while avoiding the need to train each layer in isolation, solving the design flaw.

As such, this research proposes a new multimodel RNAM capable of consistently learning nonlinear tasks in a cognitively plausible fashion. This model is composed of an unsupervised module, referred to as the Multi-Feature extraction bidirectional associative memory (MF), which contains multiple layers of interconnected FEBAMs, and a supervised module, the Bidirectional

Associative Memory (which contains a single modified BAM). It is hypothesized that this design would allow the BAM module to receive linearly separable representations generated by the MF. Consequently, this combined architecture, MF-BAM, would enable the model to find a solution through self-governed dynamics and in a consistent fashion. Most importantly, the MF-BAM would be capable of learning nonlinear tasks while keeping its internal structure consistent throughout its composing networks, e.g., same learning rule, transmission function, etc., making it a more cognitively accurate model.

Furthermore, like the layered organization of biological neurons, e.g., hypercolumns and macrocolumns in the cortex (Wang & Cui, 2018), only the number of neurons in each layer (number of units in each network) and the number of layers (number of networks) of the unsupervised component should have an impact on the model's behaviour. Changing the MF's structural parameters should substantially affect the generated output (T. Rolon-Mérette et al., 2019; Rolon-Merette et al., 2018). Suppose the MF can change nonlinear relationships between input and targets of a task into a linearly separable one. Then, the final BAM layer should be able to learn the task easily. Otherwise, the outputted behaviour of the MF-BAM will be linear, and the model will fail.

Various nonlinear problems were used to evaluate if the MF-BAM can consistently solve nonlinear tasks (Goodwin & Johnson-Laird, 2013; Nadler et al., 2008; Wattenmaker, 1995; Wattenmaker et al., 1986). These were the *N*-bit parity (bipolar value) and the Moons, Eclipse, and Suns tasks (continuous value). Other abilities, such as continuous online learning and multiclass (3-class Spiral task), were also investigated. Each problem was presented as a hetero-association task, where the model had to learn the appropriate input-target pairs. Additionally, the structural parameters, *number of units per layer,* and *number of unsupervised network layers* were

manipulated to understand the effects of their interaction on performance. Lastly, learning error, decision zones, and recall performance measurements were all used to assess these effects.

As such, to adequately present the MF-BAM, this paper is divided into the following sections: Model, General Methodology, Impact of the Number of Units per Layer (*u*), Impact of the Number of Unsupervised Network Layers (*l*), Impact of Manipulating the Number of Units (*u*) and Unsupervised Network Layers *(l*), Continuous Online Learning, Multiclass Nonlinear Task, General Discussion, and Conclusion.

3.2. Model

We introduce the MF-BAM, a Bidirectional Associative Memory neural network consisting of multiple unsupervised network layers (ULs) and a single supervised network layer (SL). The ULs are based on the FEBAM architecture (Chartier et al., 2007), while the SL consists of a modified BAM architecture (Chartier & Boukadoum, 2006a, 2006b). The MF-BAM functions by associating inputs to their corresponding targets in a bidirectional fashion via multiple transformations through its ULs. For a given input pattern, **p,** learning begins in the MF, where an initial unsupervised network layer (UL_1) learns to generate a first hidden representation, $\mathbf{h_1}$. This representation is then passed to the second unsupervised network layer (UL_2), which will learn to generate a second hidden representation, $\mathbf{h_2}$. This process is repeated for *l* number of unsupervised network layers until the deepest hidden representation, $\mathbf{h_l}$, is generated. Once the entirety of the MF has learned to produce a consistent $\mathbf{h_l}$ representation, it is then sent to the BAM component (SL), which will learn to associate $\mathbf{h_l}$ with its corresponding output target, **o**. Fig. 2 displays the overall architecture. The architecture, transmission function, and learning rule are presented in the following sections to understand further how the MF-BAM operates.

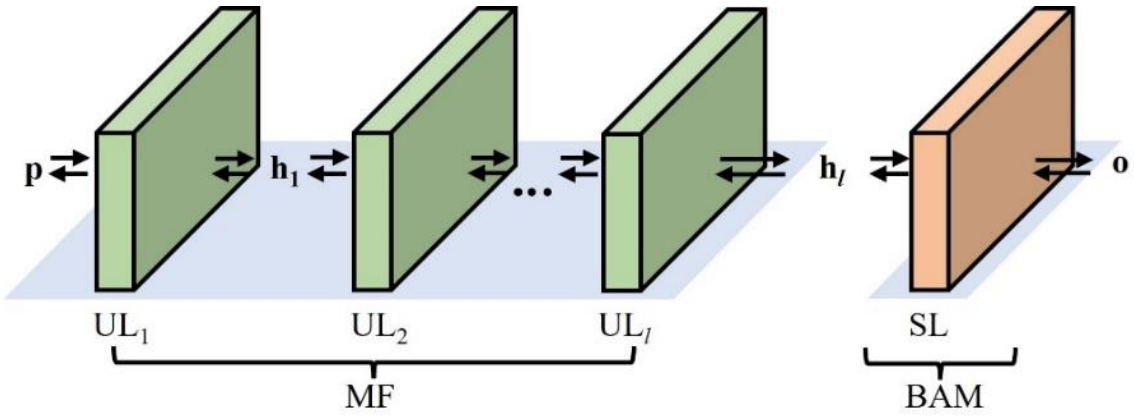

Fig. 3.2. MF-BAM architecture box model.

3. 2. 1. Architecture

The MF-BAM's architecture consists of multiple networks connected in a bidirectional fashion. The model is divided into two components: the Multi-Feature extraction bidirectional associative memory (MF), which contains various ULs, and the Bidirectional Associative Memory (BAM), which encompasses a single SL. Both the ULs and SL have two sublayers (input layer **x** and output layer **y**) of interconnected units in a bidirectional fashion. They are based on the modified version of the BAM. Contrary to Kosko's original BAM (1988), the UL and SL are connected via the independent weight matrices **W** and **V**. Due to the recurrent nature of the UL and SL, these sublayers return information to each other. As such, **W**'s shape is given by **y**'s and **x**'s dimensionality and **V** by **x**'s and **y**'s (i.e., if **x** has a dimensionality of *m* and **y** of *n,* **W** becomes a *n* x *m* matrix and **V** a *m* x *n*). As shown in Fig. 3.3, the only difference between the UL and SL is their number of external input connections. The SL has two sets of external connections and receives two inputs, $\mathbf{x}_{[0]}$ and $\mathbf{y}_{[0]}$. On the other hand, the UL only has one set and receives one input, $\mathbf{x}_{[0]}$, and generates its outputs, $\mathbf{y}_{[c]}$. This means that the number of y-units (i.e., the dimensionality of the generated hidden representations) can be manipulated for the UL exclusively.

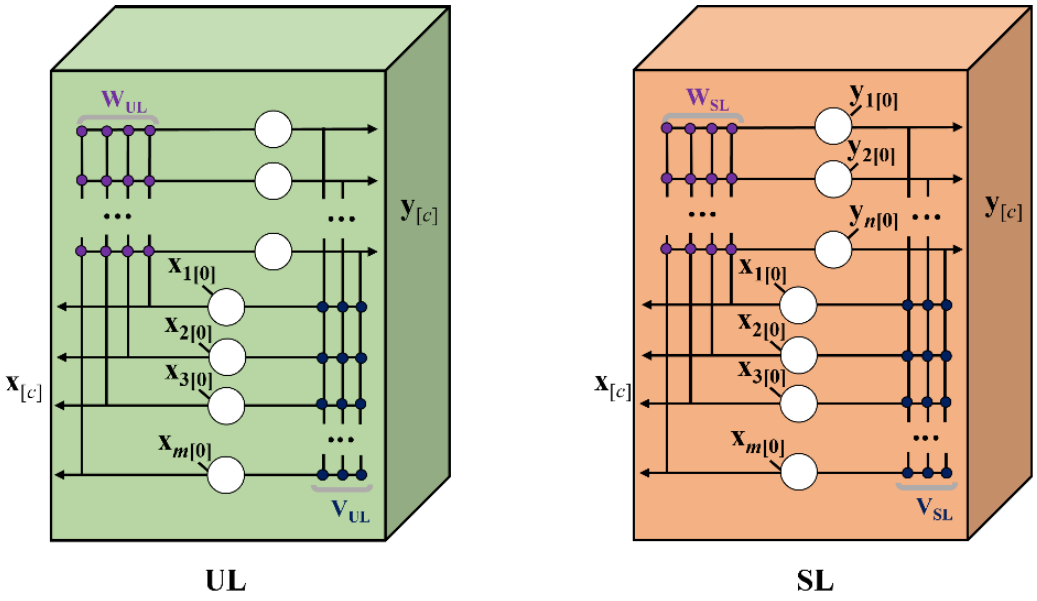

Fig. 3.3. The unsupervised (UL) and supervised (SL) network's architectures.

Since the MF-BAM is composed of two networks of the same nature, it uses the same transmission function and learning rules for all its network layers. Thus, any change in the behaviour of the MF-BAM comes only with modifications to its architecture through two parameters: the number of y-units in each UL (i.e., number of units per layer, denoted by *u*) and the number of ULs used (i.e., number of unsupervised network layers, represented by *l*). Fig. 3. 4 shows these two parameters and how they shape the overall architecture of the MF.

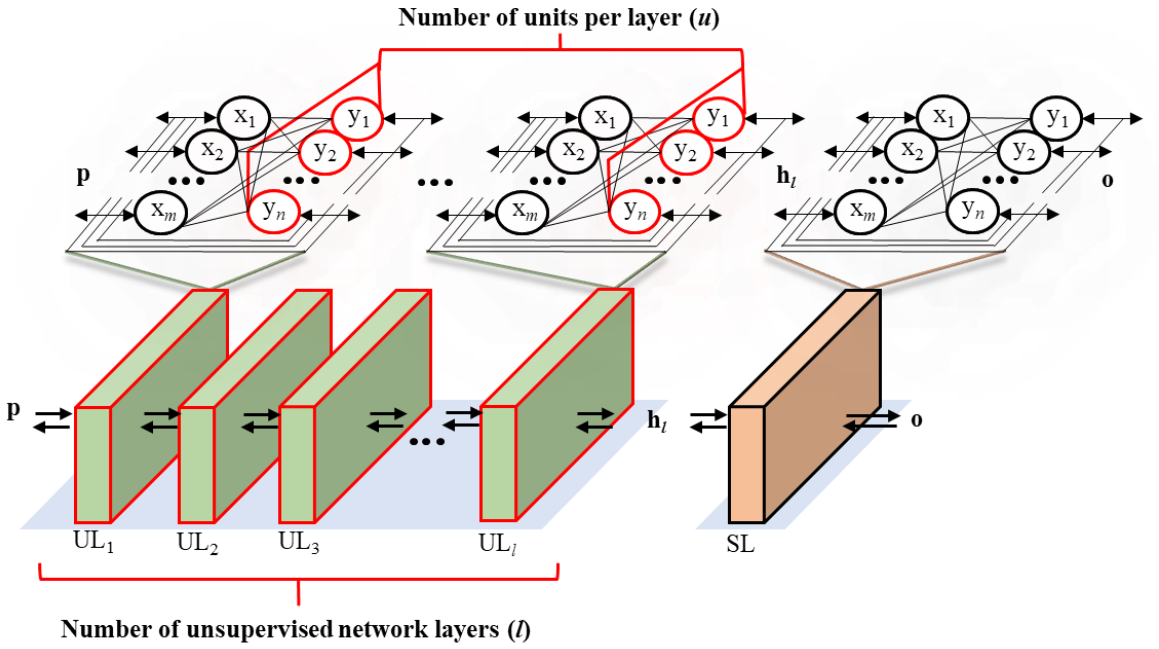

Fig. 3.4. The u (number of units per layer) and l (number of unsupervised network layers) parameters of the MF component in the MF-BAM.

3. 2. 2. Transmission Function

The MF-BAM uses a cubic transmission function with saturating limits at ±1, defined by equations (3.1a) and (3.1b):

$$(3.1a)\ \forall i, \dots, n, \mathbf{y}_{i[c+1]} = f(\mathbf{a}_{i[c]}) = \begin{cases} 1, if\ \mathbf{a}_{i[c]} > 1 \\ -1, if\ \mathbf{a}_{i[c]} < -1 \\ (\delta + 1)\mathbf{a}_{i[c]} - \delta \mathbf{a}_{i[c]}^3, Else \end{cases}$$

$$(3.1b)\ \forall i, \dots, m, \mathbf{x}_{i[c+1]} = f(\mathbf{b}_{i[c]}) = \begin{cases} 1, if\ \mathbf{b}_{i[c]} > 1 \\ -1, if\ \mathbf{b}_{i[c]} < -1 \\ (\delta + 1)\mathbf{b}_{i[c]} - \delta \mathbf{b}_{i[c]}^3, Else \end{cases}$$

where *i* is the index unit, *c* the iteration cycles index, δ a general transmission parameter, *m* and *n* the number of units in each sublayer **x** and **y**, respectively, and **a** and **b** their activations. The activations are obtained following equations (3.2a) and (3.2b):

$$(3.2a)\ \mathbf{a}_{[c]} = \mathbf{W}\mathbf{x}_{[c]}$$

$$(3.2b)\ \mathbf{b}_{[c]} = \mathbf{V}\mathbf{y}_{[c]}$$

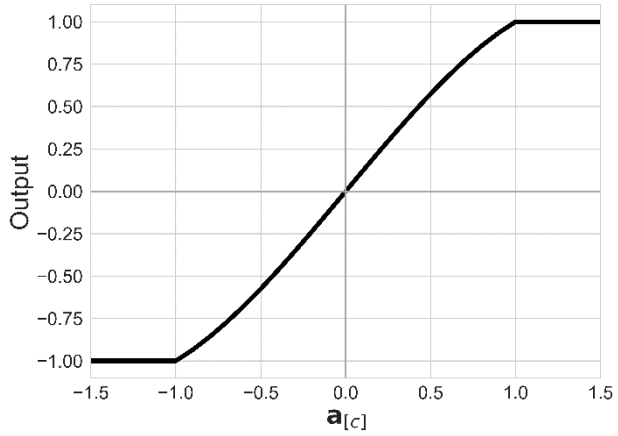

Fig. 3.5. The MF-BAM's transmission function for δ = 0.2.

3. 2. 3. Learning rule

The MF-BAM uses a Hebbian/anti-Hebbian learning rule to modify its weight connections according to equation (3.3a) and (3.3b):

$$(3.3a)\quad \mathbf{W}_{[k+1]} = \mathbf{W}_{[k]} + \eta\left(\mathbf{y}_{[0]} - \mathbf{y}_{[c]}\right)\left(\mathbf{x}_{[0]} + \mathbf{x}_{[c]}\right)^{\mathrm{T}}$$

$$(3.3b)\quad \mathbf{V}_{[k+1]} = \mathbf{V}_{[k]} + \eta\left(\mathbf{x}_{[0]} - \mathbf{x}_{[c]}\right)\left(\mathbf{y}_{[0]} + \mathbf{y}_{[c]}\right)^{\mathrm{T}}$$

where η is the learning parameter, *k* is a given learning trial, $\mathbf{x}_{[0]}$ and $\mathbf{y}_{[0]}$ are the initial incoming patterns presented to the **W** and **V** matrices and $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$ are their respective reconstructed patterns at the iteration cycle *c*. Fig. 3.6a) shows an example for a UL of how an incoming signal will go through **W**, **V,** and back to **W** to obtain the reconstructed generated pattern $\mathbf{y}_{[1]}$ and mark the end of an iteration cycle. Furthermore, it also shows how this process can continue for *c* number

of iteration cycles. For the SL, the iteration cycle is shortened as the incoming pattern $\mathbf{x}_{[0]}$ traverses $\mathbf{W}$, and the incoming "teacher" pattern $\mathbf{y}_{[0]}$ passes $\mathbf{V}$ simultaneously to produce the reconstructed "teacher" pattern $\mathbf{y}_{[1]}$ and the reconstructed incoming patterns $\mathbf{x}_{[1]}$, respectfully. Likewise, $\mathbf{x}_{[1]}$ and $\mathbf{y}_{[1]}$ can be sent back for a c number of iteration cycles to give $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$. Fig. 3.6b) shows this process for the SL, where the black lines represent the process for an iteration cycle of 1, and the dashed lines show how information can circulate back in $\mathbf{W}$ and $\mathbf{V}$ for c iteration cycles.

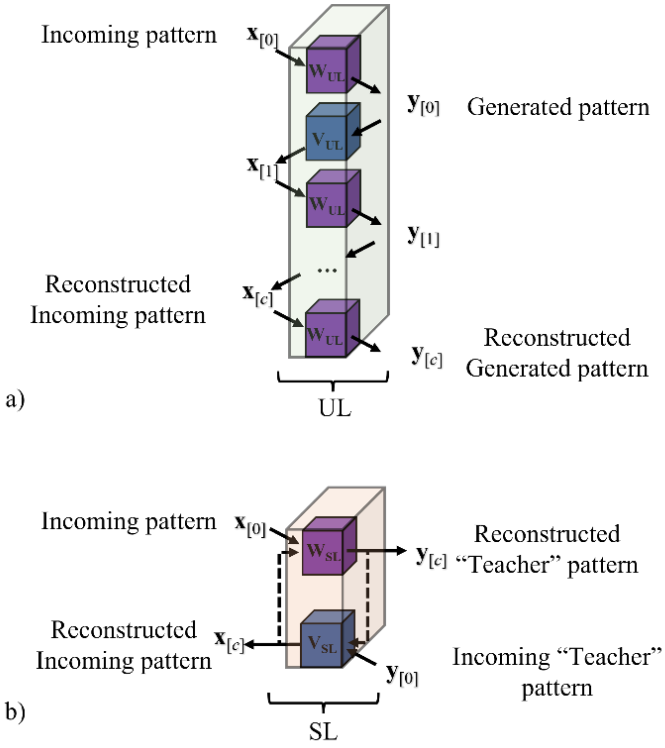

Fig. 3.6. Box diagram of a) UL's iterative cycle and b) SL's iterative cycle.

It is important to note that the Hebbian/anti-Hebbian learning rule (equations (3.3a) and (3.3b)) can be driven by the incoming pattern, $\mathbf{x}_{[0]}$ and/or $\mathbf{y}_{[0]}$, and their reconstructed version at any c iteration cycle. As such, once all desired terms are acquired ($\mathbf{x}_{[0]}$, $\mathbf{x}_{[c]}$, $\mathbf{y}_{[0]}$, and $\mathbf{y}_{[c]}$) in a network layer, a weight update for both $\mathbf{W}$ and $\mathbf{V}$ can occur, resulting in the learning trial, k, incrementing by one and the c variable resetting. Furthermore, during learning, weight update is said to have stabilized when $\mathbf{x}_{[0]} = \mathbf{x}_{[c]}$ or $\mathbf{y}_{[0]} = \mathbf{y}_{[c]}$ (i.e., when the input layer and output layer have become

constant in their activation). This means that despite continuing the weight update, both **W** and **V** remain centered at a particular set of values which represents minima in the solution space. In other words, a network layer's stabilization means that its internal dynamics have produced stable attractors representing the learned patterns. This stabilization can only be guaranteed if the learning parameter (η) is small and positive. Fig. 3.7a) shows an example of a UL and 3.7b) a SL, stabilizing during learning to minima when $\mathbf{x}_{[0]} = \mathbf{x}_{[c]}$ (Red) and $\mathbf{y}_{[0]} = \mathbf{y}_{[c]}$ (Blue) for $c = 1$.

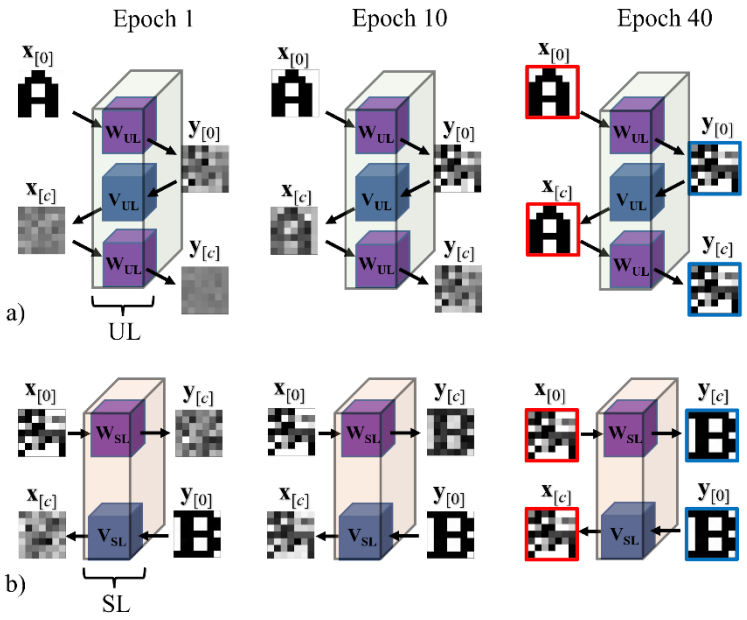

Fig. 3.7. Example of a) UL's and b) SL's learning for an iterative cycle of $c = 1$ and for 40 epochs on a simple association task.

3. 2. 4. MF-BAM's learning procedure

To better understand how the MF's architecture impacted the BAM's behaviour, each module was trained individually to highlight this effect. As such, in the MF-BAM, information cycling begins with its MF component. Each UL of the MF functions the same as described in section 3.2.3., where an input pattern arrives at the input layer, denoted as $\mathbf{x}_{\mathbf{UL}[0]}$, and traverses through the weight matrix $\mathbf{W}_{\mathbf{UL}}$ to generate $\mathbf{y}_{\mathbf{UL}[0]}$ at the output layer. This generated pattern $\mathbf{y}_{\mathbf{UL}[0]}$ is then sent back to the input layer via the $\mathbf{V}_{\mathbf{UL}}$ weight matrix to reconstruct the incoming pattern, denoted as $\mathbf{x}_{\mathbf{UL}[c]}$.

The first iterative cycle is complete once $\mathbf{x}_{\mathbf{UL}[c]}$ is sent back to the $\mathbf{W}_{\mathbf{UL}}$ matrix to generate $\mathbf{y}_{\mathbf{UL}[c]}$. Once all four terms, $\mathbf{x}_{\mathbf{UL}[0]}$, $\mathbf{y}_{\mathbf{UL}[0]}$, $\mathbf{x}_{\mathbf{UL}[c]}$, and $\mathbf{y}_{\mathbf{UL}[c]}$, are obtained for a given UL, a weight update can occur as per equations (3.3a) and (3.3b). The $\mathbf{y}_{\mathbf{UL}[c]}$ can then be sent to the following UL as input, and the c can be reset for the following incoming signal. This initiates the same process for the subsequent UL. In other words, when an initial pattern $\mathbf{p}$ arrives in the MF component, it triggers a cascade of this cyclic process until the final network layer, UL_l, produces a stable $\mathbf{y}_{\mathbf{UL}l[c]}$. For simplicity, each UL's $\mathbf{y}_{\mathbf{UL}[c]}$ will be referred to as a hidden generated representation of that network layer, denoted by $\mathbf{h}$. Thus, UL_1 will generate a $\mathbf{h}_1$, UL_2, a $\mathbf{h}_2$, …, and UL_l, a $\mathbf{h}_l$. Fig 3.8 shows this process for the MF and its bidirectional nature.

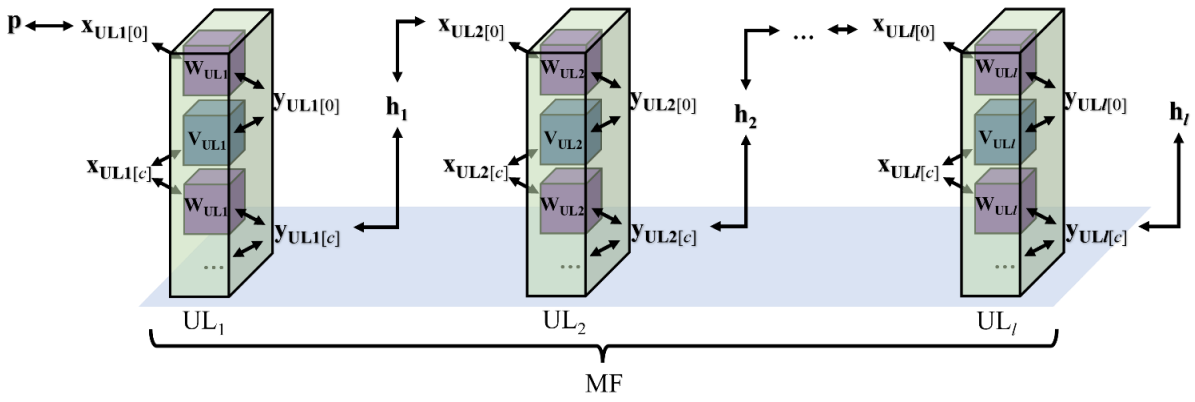

Fig. 3.8. Box diagram of the MF's learning process.

In sum, in the MF, when an input pattern arrives at a UL, the signal can circulate between the $\mathbf{W}_{\mathbf{UL}}$ and $\mathbf{V}_{\mathbf{UL}}$ matrices for a c number of iteration cycles. Once all four desired terms are obtained ($\mathbf{x}_{\mathbf{UL}[0]}$, $\mathbf{y}_{\mathbf{UL}[0]}$, $\mathbf{x}_{\mathbf{UL}[c]}$, $\mathbf{y}_{\mathbf{UL}\,[c]}$), a weight update occurs, c is reset, and a learning trial (k) has passed for that UL. Once all input patterns have been individually presented, it is said that one epoch (e) has passed. This learning process can continue until a maximum number of epochs has been reached or if the entirety of the MF has stabilized. In other words, until the maximum number of repetitions has passed or when weight updates in every UL have stabilized.

Because the MF doesn't require error to be backpropagated, weight updates can occur independently for each UL. Furthermore, since all ULs are sequentially linked, a cascade of weight update occurs, and a stable final behaviour $\mathbf{h}_l$ is obtainable from UL_l once all previous network layers have stabilized. This independence also means that each UL can continue its weight updates even if it has stabilized. In other words, if the input signal is constant and a UL has stabilized, no significant change should occur while maintaining the updates. Fig. 3.9 shows the MF's learning process and how its deeper networks require more time to stabilize and produce a consistent hidden representation of a solution on a simple association task. This is illustrated in Fig.3.9, where, as an individual UL stabilizes, their generated representations even outs, as shown by the contour colour shifting from blue (initial representation) to red (stable solution).

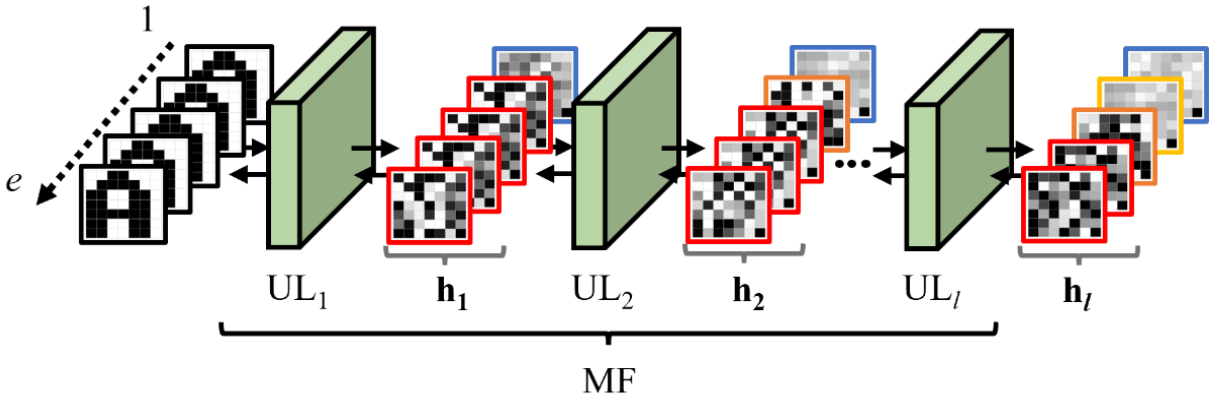

Fig. 3.9. Visual representation of MF's learning process on a simple paired association task.

Once the MF has learned to produce a stable set of generated patterns $\mathbf{h}_l$ from its final UL, these are then given to the BAM component for learning. From its SL, this component can start associating the set of $\mathbf{h}_l$ with their corresponding set of output targets, **o**. As previously described in section 3.2.3., this learning process is shortened compared to the MF. As $\mathbf{h}_l$ arrives as the incoming pattern in the input layer $\mathbf{x}_{\mathbf{SL}[0]}$ and its corresponding target **o** as the teacher pattern in

the output layer $\mathbf{y}_{SL[0]}$, they traverse $\mathbf{W}_{SL}$ and $\mathbf{V}_{SL}$ simultaneously to give $\mathbf{y}_{SL[c]}$ and $\mathbf{x}_{SL[c]}$, respectively. Once all four terms are obtained ($\mathbf{x}_{SL[0]}$, $\mathbf{y}_{SL[0]}$, $\mathbf{x}_{SL[c]}$, and $\mathbf{y}_{SL[c]}$), a weight update can occur, and the learning trial (k) is increased by one, and c is reset. Once all $\mathbf{h}_l$ and their corresponding $\mathbf{o}$ have been individually presented, it is said that one epoch (e) has passed. This learning process can continue until a maximum number of epochs has been reached or if the SL has stabilized. Fig. 3.10 shows this process for the BAM component where the solid lines represent an iteration cycle c = 1, and the dashed lines show how information can circulate back for several c iteration cycles.

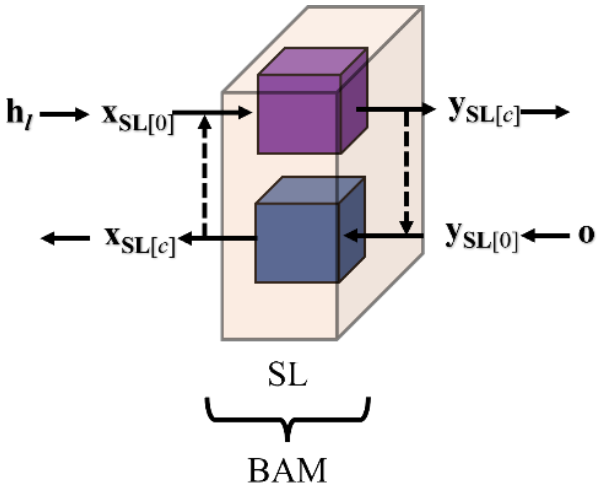

Fig. 3.10. Box diagram of the BAM component's learning process

Conjunctively, the MF and BAM component function by having the prior learn to produce a set of generated patterns that the latter can associate with a given target. This entire rationale is shown in Fig. 3.11 for a simple association task (pair patterns A and B). As the MF receives constant information, its individual UL stabilizes to a set of weight values, producing a stable generated pattern $\mathbf{h}_l$ (red) that can then be given to the BAM to learn to pair it with B. Once learning is complete, presenting A to the MF-BAM will generate all hidden representations and the desired

output, pattern B. Furthermore, due to its bidirectional nature, giving B to the output layer of the SL can also retrieve all hidden representations and reconstruct pattern A at the input layer of UL_1.

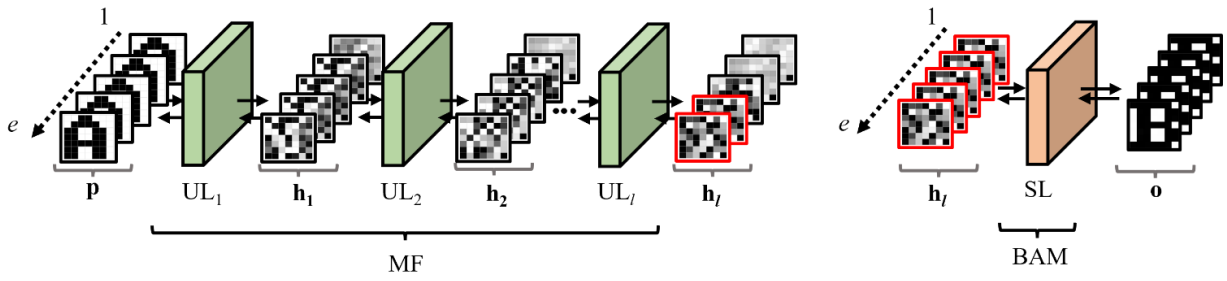

Fig. 3.11. Box diagram of the MF-BAM's learning process for a simple association task.

3.3. General Methodology

Multiple tasks were used in different simulations to evaluate the MF-BAM's learning capabilities. For all simulations, the learning and recall procedure remained identical, with the transmission function parameter (δ) set to 0.2 and the learning parameter (η) set to respect $0 < \eta < 1/max(m, n)$. The use of input bias neurons initialized to a value of 1 was added to every input layer (ULs and SL) of the MF-BAM (Aggarwal, 2018). For all network layers, weights were randomly initialized following a uniform distribution with values between -0.1 and 0.1. The number of units per layer (u) and the number of unsupervised network layers (l) were set according to the simulation condition. During the learning process, the number of cycles before the weight updates was set to $c = 1$. The MF and the BAM component stopped learning when the maximum number of epochs (e) for a given simulation was achieved. The entire model and testing protocols were created and implemented with the Python programming language installed from Anaconda (Inc., 2020) and run on a RTX3090 graphics card (Church et al., 2021; Rolon-Mérette et al., 2016).

3. 3. 1. Measurements

Three measurements were used to ensure that the MF-BAM's behaviour would be adequately evaluated: learning error, decision zones, and recall performance.

3. 3. 1. 1. Learning Error

Learning error was measured with mean squared error (MSE). For the MF component, the MSE value represented the difference between the generated patterns at interaction cycle *c* and those at *c*-1. For the BAM component, MSE described differences between the predicted output and the actual target. Thus, low MSE values (< 0.001) were sought out for both parts, as they indicated a stable generated representation (MF) and successful input-target associations (BAM).

3. 3. 1. 2. Decision Zones

For all tasks containing 2-dimensional input vectors, decision zones were generated by presenting test inputs. These test inputs were created by mapping out the 2D space between the [-1, -1] and [1, 1] coordinates in steps of 0.01. Thus, 40 401 test inputs of 2 dimensions were created. Using decision zones as a measurement can be equivalent to training the network on a subset of data and testing the model on novel instances, like what is traditionally done in train-test split protocols (Raschka, 2018). As such, this measurement allows seeing the overall classification behaviour of the model.

3. 3. 1. 3. Recall Performance

To ensure the model could learn the appropriate associations to provide a steady solution, recall performances were measured in two ways. The first was simply by recording the average recall performance (how many patterns were successfully classified) and the standard deviation. The

second was by only counting perfect scores, i.e., successes vs. failures. A step-function was applied to the SL's output to facilitate recall performance calculations, where any node values ≥ 0 would be rounded up to 1, and anything < 0 would be given a -1 value. This step function was strictly applied for recall performance calculations only.

3. 3. 2. General Learning Procedure

The learning procedure was conducted in two phases: UL learning (3.3.2.1.) and SL learning (3.3.2.2.).

3. 3. 2. 1. - UL learning

For the MF, each epoch (*e*) consisted of the following steps:

1. Creation of a list, **P**, containing all input patterns **p**.
2. Selection of a random input pattern **p** to be given to the first UL.
3. Incoming pattern for UL_l, denoted by $\mathbf{x}_{\mathbf{UL}l[0]}$.
4. Computation of the generated pattern $\mathbf{h}_l$, denoted by $\mathbf{y}_{\mathbf{UL}l\,[c]}$, according to equations (3.1) and (3.2), and as described in sections 2.3 and 2.4.
5. Computation of weight update for $\mathbf{W}_l$ and $\mathbf{V}_l$ according to equation (3.3).
6. Relaying of $\mathbf{h}_l$ to the following UL.
7. Repetition of 3-6 for the following UL until all ULs have been updated.
8. Repetition of 2-7 until all input patterns in **P** have been exhausted.

Learning continued until the *e* condition was reached for a given simulation.

3. 3. 2. 2. - SL learning

For the BAM, it was deemed that an epoch had passed after the following steps were completed:

1. Creation of a list, **HO**, containing all generated patterns from the final UL of the MF ($\mathbf{h}_l$) and their corresponding targets (**o)**.
2. Selection of a random pair of generated patterns and their corresponding targets to be given as $\mathbf{x}_{\mathbf{SL}[0]}$ and $\mathbf{y}_{\mathbf{SL}[0]}$, respectfully.
3. Computation of $\mathbf{x}_{\mathbf{SL}[c]}$ and $\mathbf{y}_{\mathbf{SL}[c]}$ according to equations (3.1) and (3.2) and as described in sections 3.2.3 and 3.2.4.
4. Weight update according to equation (3.3).
5. Repetition of 2-4 until all pairs of patterns in **HO** have been exhausted.

Learning continued until the *e* condition was reached for a given simulation.

3. 3. 3. General Recall Procedure

The recall procedure for the MF-BAM was performed in the following fashion:

1. Selection of a desired input pattern to be given to the first UL as $\mathbf{x}_{\mathbf{UL}[0]}$.
2. Computation of $\mathbf{y}_{\mathbf{UL}[c]}$ according to equations (3.1) and (3.2).
3. Relaying $\mathbf{y}_{\mathbf{UL}[c]}$ to the following UL to be used as $\mathbf{x}_{\mathbf{UL}l[0]}$.
4. Repetition of 2-3 for each UL until the final generated representation is obtained, $\mathbf{y}_{\mathbf{UL}l[c]}$.
5. Relaying $\mathbf{y}_{\mathbf{UL}l[c]}$ as $\mathbf{h}_l$ to the SL to be used as $\mathbf{x}_{\mathbf{SL}[0]}$.
6. Computation of $\mathbf{y}_{\mathbf{SL}[c]}$ according to equations (3.1) and (3.2).

A comparison between $\mathbf{y}_{\mathbf{SL}[c]}$ and its original corresponding target $\mathbf{y}_{\mathbf{SL}[0]}$ was performed by following section 3.3.1.3 to determine if the model made a mistake.

3.4. Impact of the Number of Units per Layer (*u*)

In this section, the main two goals were to evaluate whether the MF-BAM could solve different nonlinear tasks and to isolate the effect of *u* (number of units per layer) on performance. As such, several nonlinear tasks separated into two types, bipolar (*N*-bit) and continuous (Moons, Eclipse, and Suns), were used. Furthermore, the number of units per layer (*u*) was systemically varied while the number of unsupervised network layers (*l*) and epochs (*e*) were kept constant.

3. 4. 1. The *N*-bit Parity Problem

The MF-BAM was initially tested on the *N*-bit parity problem, a classical and popular benchmark for ANNs. The task involves determining if a sequence of binary digits (consisting of 0s and 1s) contains an odd or even number of 1s. The complexity of the challenge increases as the sequence length, represented by *N*, increases. Notably, in the bipolar version, the solution involves counting the number of -1s instead of 1s and determining if it's odd or even.

3. 4. 1. 1. Methodology

The *N*-bit parity problem was implemented by creating bipolar input pattern vectors, **p**, containing values of -1 or 1, of *N* dimensionality (i.e., $\mathbf{p} = [p_1, p_2, \ldots, p_N]$). The total number of inputs, **P**, determined by 2^N, was created by mapping the bipolar input space. The target for each input was determined by equation 3.4 and was represented as a 1-dimensional vector containing either the values of -1 or 1, signifying odd or even counts of -1s, respectively.

$$(3.4)\ For\ \mathbf{p}\ in\ \mathbf{P}, \varphi(\mathbf{p}) = \begin{cases} 1, if\ \prod_{k=1}^{N} \mathbf{p}_k\ \ is\ positive \\ -1, if\ \prod_{k=1}^{N} \mathbf{p}_k\ \ is\ negative \end{cases}$$

To evaluate different levels of difficulties, *N* was varied from 2 to 6. Higher *N* values were also studied but were omitted from this section since the results followed the same trends. An example of the 2-bit and 3-bit can be seen in Table 3.1, where the inputs and their corresponding targets are shown for both tasks.

Table 3.1. Inputs and corresponding targets for the 2- and 3-bit tasks.

2-bit			**3-bit**							
Inputs		Targets	Inputs			Targets	Inputs			Targets
1	1	1	1	1	1	1	-1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	1	-1	1
1	-1	-1	1	1	-1	-1	-1	-1	1	1
-1	1	-1	1	-1	-1	1	-1	1	1	-1

For these simulations, the model's architecture was set to $l = 4$, and u varied from 1 to 150. The learning and recall followed the procedure in section 3. Learning for the MF and BAM was set to 2500 epochs (*e*) each. The task was repeated 100 times for every bit size for reliability assessment. Average learning times, decision boundaries, and total recall performances were also reported.

3. 4. 1. 3. Results

The average number of epochs and mean squared error (MSE) for the 2-bit task are shown in Fig. 3.12(a). In general, MF learning was characterized by a decrease in error (MSE) over time (number of epochs), stabilizing at low MSE values (< 0.001). However, when looking at individual ULs' MSE, spikes in error are observed when previous ULs stabilize. Nevertheless, after these spikes, the MSE drops to low values. For example, when $u = 100$, the MSE of the second UL (orange) spiked once the first UL (blue) stabilized but decreased again to low values.

For the BAM component, learning was also characterized by a decrease in MSE as the number of epochs increased. However, the BAM did not always stabilize at low MSE values. For the 3 and 5 units per layer conditions, the BAM stabilized on high MSE values (≈ 1.75 and ≈ 0.75, respectively), which hinted that it did not successfully learn the task. However, for other conditions ($u \geq 10$), it stabilized at an MSE value lower than 0.001.

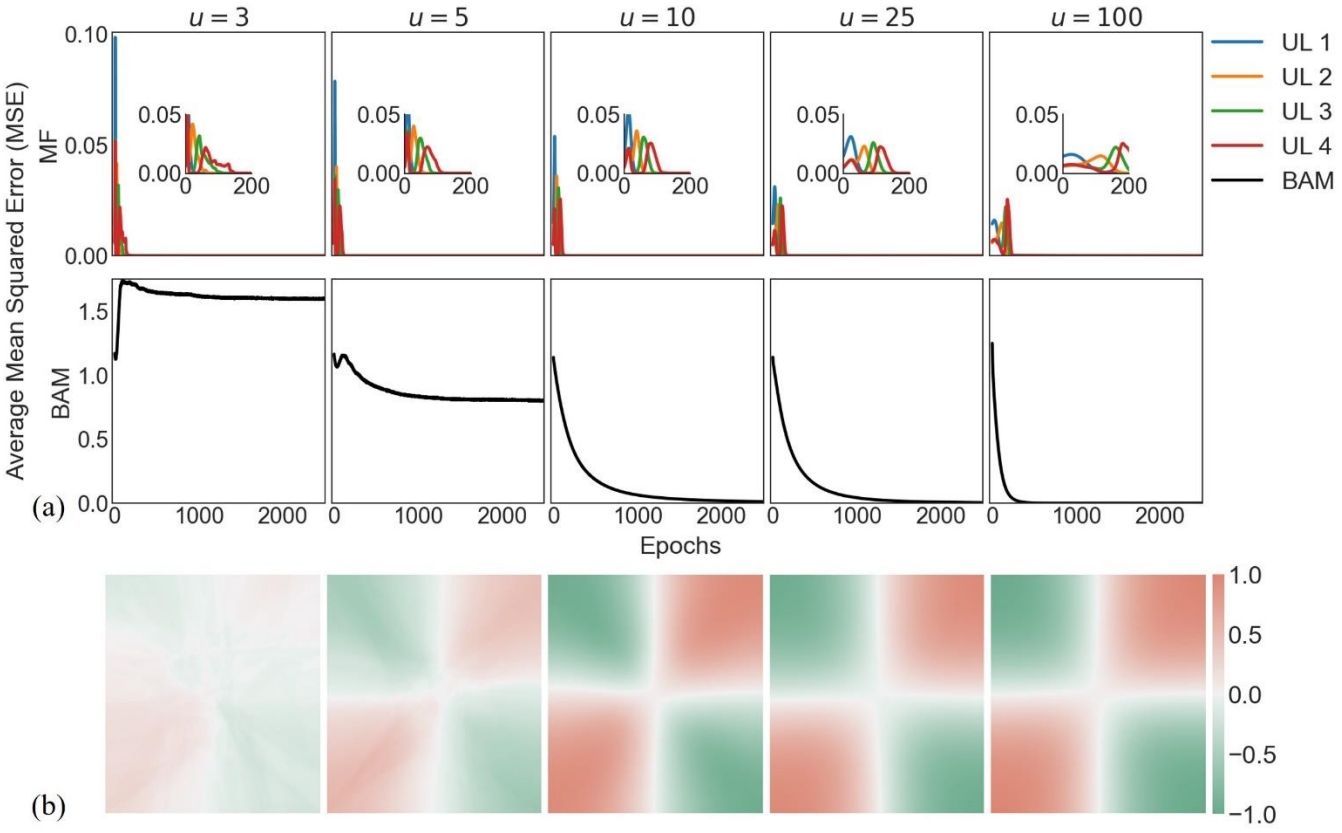

Fig. 3.12. (a) Average mean squared error during learning for both the MF and BAM component for the 2-bit task under different units per layer (u) conditions. (b) Average decision boundaries for the 2-bit task under the same conditions.

Results of average decision zones after 100 trials for the 2-bit task are shown in Fig. 3.12(b). For u = 3, there was no clear observable decision boundary, confirming the failure of the MF-BAM to learn the 2-bit task as suggested by the high values of MSE. However, at u = 5, the decision boundary containing the shape of the appropriate solution began to emerge, albeit with prediction

values very close to 0, which coincided with the BAM's high error. Increasing *u* resulted in these values becoming increasingly closer or equal to –1 and 1, ultimately giving rise to the decision zones that best encapsulate the solution to the task.

Performances for all *N*-bit tasks across the 100 trials for each *u* condition are shown in Fig. 13. For each version of the *N*-bit, 100% recall performance was achieved once *u* was large enough. For instance, perfect performance was observed at $u \geq 10$ for the 2-bit task but at $u \geq 30$ for the 3-bit. Generally, the MF-BAM would begin to correctly learn the task at a given number of units per layer and quickly reach perfect performance with only a few additional units per layer.

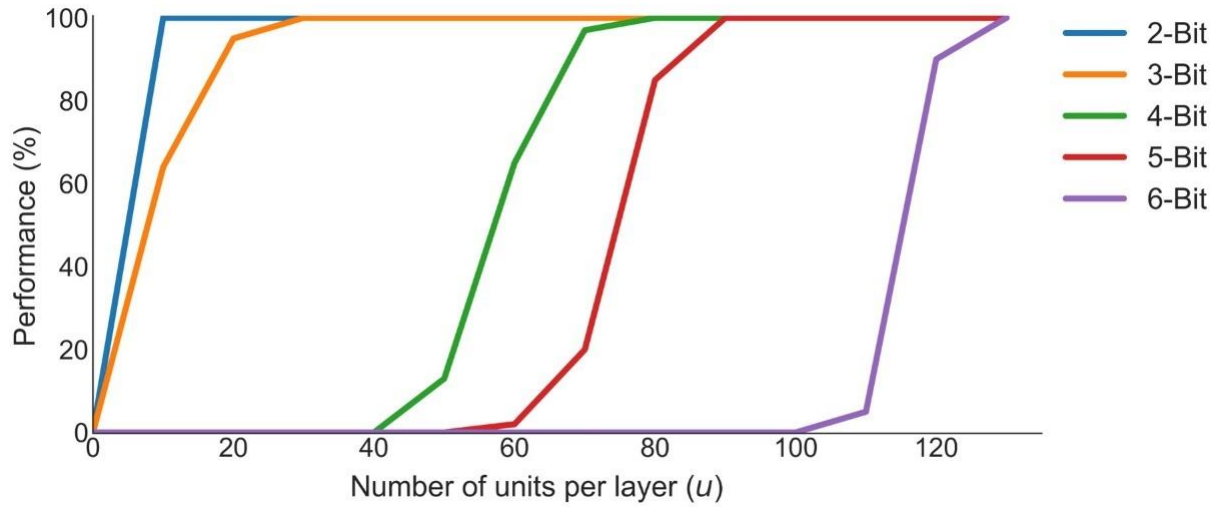

Fig. 3.13. Recall performance of the MF-BAM for the *N*-bit task in function to the number of units per layer (*u*).

3. 4. 2. Continuous Value Tasks

The MF-BAM was also tested on other sets of popular nonlinear tasks. However, the inputs now consisted of continuous values rather than bipolar ones. More specifically, the Moons, the Eclipse, and the Suns tasks were used. Like section 3.4.1, the number of units per layer was manipulated, while *l* and *e* were kept constant.

3. 4. 2. 1. Methodology

To create the Moons, Eclipse, and Suns tasks, 40 two-dimensional input vectors were generated to represent a Cartesian plane's x and y coordinates. These inputs were divided into two classes and assigned one-dimensional target vectors of value -1 or 1. The Moons and Eclipse were generated using the Scikit Learn library in Python (https://scikit-learn.org/stable/; (Buitinck et al., 2013; Pedregosa et al., 2011) with the *make_moons* and *make_circles* functions, respectively. The parameters used for both functions were *n_sample* = 40 and *noise* = 0, and the additional *factor* parameter was set to 0.7 for the Eclipse. The Suns task was generated through a custom function, which spawned two circles, one of which was 0.5 times smaller than the other, and its center offset to the left by a factor of 1.2. The input vector values for each task were scaled between -1 and 1, and the goal was to associate them with their respective target vectors. All three tasks are displayed in Fig. 3.17. Although larger data sets were also examined, their results were not included as they mirrored the trend in the presented data.

To evaluate the effect of *u*, the value of *l* was fixed at 4, and five conditions were created where *u* ranged from 100 to 500 in increments of 100. The learning and recall process followed the procedure described in section 3. For every condition, *e* was set to 2500 for both the MF and BAM components. To assess reliability, the learning and recall process was conducted for 100 trials. The average learning and recall performance and decision zones for all trials were reported.

3. 4. 2. 2. Results

The MF and BAM's average number of epochs and MSE are shown in Figs. 3.14, 3.15, and 3.16 for the Moons, Eclipse, and Suns task, respectively. In each task, the learning error of the MF component followed the same sequence found in the *N*-bit task. As each previous UL stabilized, a minor blip in MSE was observed for the following ULs and afterward quickly decreased and

stabilized at low values (<0.001). In the BAM component, for the Moons task, the MSE error was higher than 0.0001 when the number of units per layer was 300 or less, suggesting unsuccessful learning of the input-target pairs.

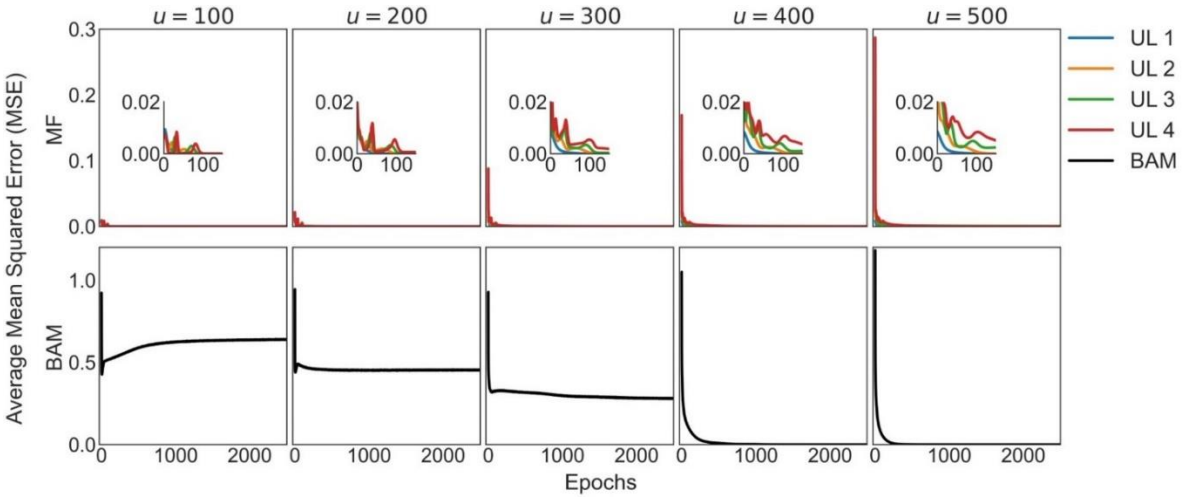

Fig. 3.14. Average mean squared error during learning for both the MF and BAM components for the Moons task in function of the number of units (u) per layer.

For the Eclipse task, the BAM's MSE values were below 0.001 when $u \geq 200$. However, in the 100 units per layer condition, the observed trend indicates that with more learning time (more epochs), the BAM would reach lower MSE values.

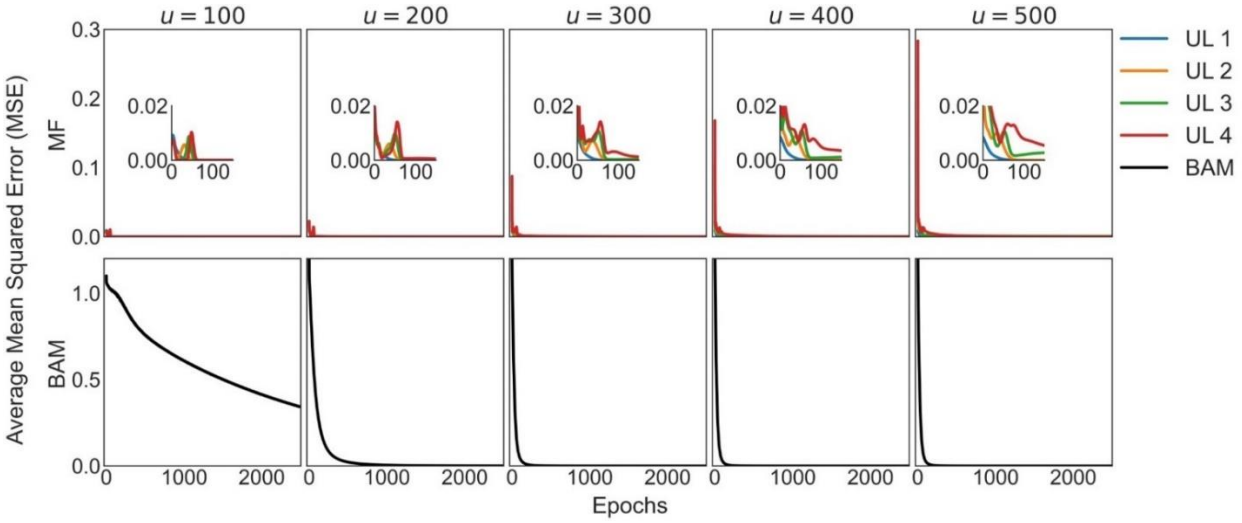

Fig. 3.15. Average mean squared error during learning for both the MF and BAM components for the Eclipse task in function of the number of units (*u*) per layer.

For the Suns task, the BAM's MSE values were higher than 0.001 regardless of the units per layer condition. This is shown in Fig. 3.16.

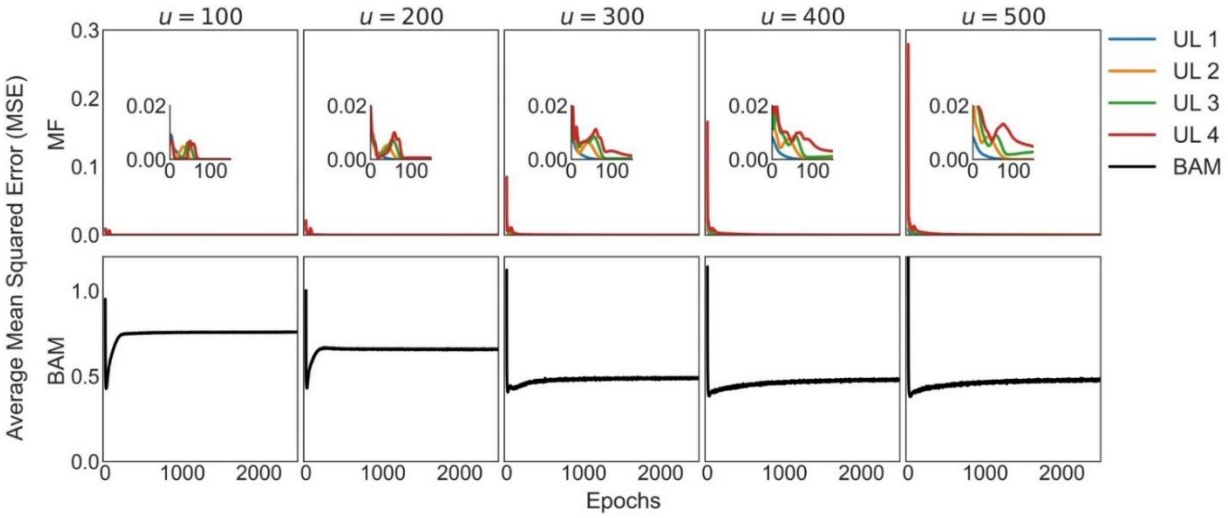

Fig. 3.16. Average mean squared error during learning for both the MF and BAM components for the Suns task in function of the number of units (*u*) per layer.

Fig. 3.17 shows the MF-BAM's decision zones for all three tasks. Regarding the Moons, with $u < 200$, the MF-BAM could only generate a quasilinear decision boundary. As u increased, the more nonlinear the separation became. With $u \geq 400$ units, the decision boundary generated by the MF-BAM allowed perfect classification. Regarding the Eclipse task, it was much easier for the model to achieve an ideal classification ($u \geq 200$ units). Finally, the Suns task showed that as u increased, the decision boundary was more and more curved. Still, the model couldn't learn to associate the three patterns inside the inner circle with the appropriate target, as shown by the misclassification zone in this area.

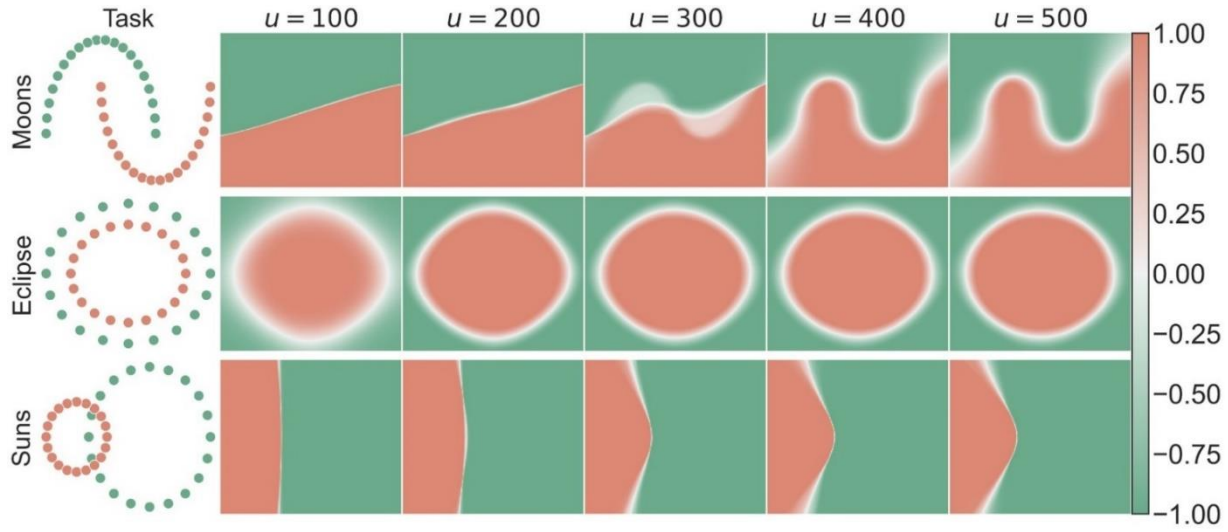

Fig. 3.17. Average decision boundaries of the MF-BAM for the Moons, Eclipse, and Suns task in function of units (u) per layer.

Average recall performances and standard deviations for all three tasks are presented in Table 3.2. For the Moons and Eclipse, average recall performance improved as u increased. A perfect recall was achieved at $u \geq 400$ for the Moons and $u \geq 200$ for the Eclipse. Further increasing of u had no effect on performance or consistency (confirmed by standard deviation = 0). Concerning the Suns task, no perfect recall was observed under any u value, with the highest average performance being

87.45% at the 300 units per layer condition. Standard deviation increased as *u* increased, ranging from 1.82 for *u* = 100 to 3.76 for *u* = 500.

Table 3.2. Average recall performance and standard deviation for all three nonlinear tasks.

		Number of units per layer (*u*)				
		100	200	300	400	500
Moons	Mean	83.05	87.85	92.20	100.00	100.00
	Std.	2.11	2.03	5.37	0.00	0.00
Eclipse	Mean	92.08	100.00	100.00	100.00	100.00
	Std.	9.70	0.00	0.00	0.00	0.00
Suns	Mean	80.12	82.42	87.45	86.40	86.80

3. 4. 3. Impact of the Number of Units per Layer Discussion

In section 4, the *u* parameter was manipulated while keeping *l* constant at 4, and it was first revealed that the MF-BAM could consistently learn nonlinear tasks. As shown in sections 3.4.1 and 3.4.2, 100% performance was always achieved in the *N*-bit, eclipse, and moons tasks. Furthermore, increasing *u* produced lower learning errors (Figs. 3.12(a), 3.14, 3.15, and 3.16), more explicit bipolar decision boundaries meaning improved classification behaviour (Figs. 3.12(b) and 3.17), and better recall performance (Fig. 3.13 and Table 3.2). If *u* was too low, the SL would struggle to learn a task consistently or at all, leading to high learning errors, poor performance, and unclear decision boundaries. For example, in a condition with *u* = 5 in the 2-bit task, the SL presented high MSE values (> 0.7) and only learned the task half the time (performance ≈ 50%). On the flip side, when *u* = 100, the supervised network layer stabilized to low MSE values > 0.001 and achieved consistent 100% performance. When looking at the model's

classification behaviour in Fig. 3.12(b), the appropriate decision boundaries showing four regions in the corners of the input space were not present when $u = 3$ and were still unclear when $u = 5$. However, these increased in clarity as the number of units rose. In these cases, predictions for unseen inputs were higher nearer the corner than the center. Inputs very close to the Cartesian axes predict a specific class membership of around 0.

Moreover, the minimum value of u required to solve a task varied from task to task. In the N-bit tasks, perfect performance was achieved all the time when u was > 10 for 2-bit, > 30 for 3-bit, > 65 for 4-bit, > 85 for 5-bit, and > 125 for 6-bit. The higher the number of bits, the higher the number and size of inputs required for the model to learn the task. Interestingly, 200 and 400 units were needed to achieve steady, perfect performance for the Eclipse and Moons tasks, respectively. The inputs' dimensionality and quantity remained the same in both tasks, but the complexity of the decision boundaries differed. The decision boundary for the Eclipse task was circular, while a more complex boundary was required for the Moons task. Hence, increasing u also assisted in learning tasks with more complicated arrangements of inputs.

However, increasing u alone was not enough to achieve perfect learning of the Suns task. This was indicated by the SL's MSE values never reaching zero (as seen in Fig. 3.16) and the performance plateauing at around 85% (as shown in Table 2). This can be explained by looking at the decision boundaries illustrated in Fig. 3.17. While the general shape of the task was achieved, the three exceptions (green points inside the red circle) were not accounted for, even with 500 units. These results become interesting when contrasted with what is traditionally sought with multilayered ANNs. With such models, a train-test split protocol is usually used to avoid overfitting a task where a subset of the data is withheld during learning to ensure that the network's

solution remains general. However, the results of the MF-BAM showed that despite presenting all the training data, the model did not overfit the task, and a general solution was obtained.

To sum up, results from section 3.4 have shown that by manipulating the number of units per layer in an architecture composed of four ULs, it was possible to teach the BAM module to classify the different nonlinear tasks regardless of them containing inputs with bipolar or continuous values. However, the model didn't thoroughly learn the Suns task and could only produce a generalized solution, not an overlearning (overfitting) one. To successfully solve the Suns task, it is required that the MF-BAM displays some sort of overlearning. This was investigated in the next section by manipulating the *l* parameter rather than *u*.

3. 5. Impact of the Number of Unsupervised Network Layers (*l*)

In the previous section, a fixed number of unsupervised network layers (*l*) was used while the number of units per layer (*u*) was manipulated. This section investigated the effect of varying *l* while controlling for *u* and *e*. The Suns task was revisited to show if the MF-BAM could solve it by displaying some overlearning. Furthermore, longer simulation times for the MF were used to investigate if each UL would maintain its stability while continuing weight updates.

3. 5. 1. Methodology

The Suns task previously described in 4.2. was used. Here, *l* varied from 1 to 13, whereas *u* was fixed to 500 units. Learning and recall performance and decision zones were collected as described in section 3, and *e* was set to 25000 for the MF and 2500 for the BAM. Learning was repeated 100 times per condition for reliability purposes, and average results were reported.

3. 5. 2. Results

Fig. 3.18(a) shows the average MSE for both the MF and BAM. For the MF component, as shown in the previous section, while the number of learning epochs increased, small increases in error were seen for all ULs once a previous layer stabilized. This stability was maintained while the learning continued. For the BAM component, average MSE decreased until it stabilized. Results showed that with an MF of 1 layer, the BAM's error increased with the number of epochs. With an MF of 4 or 7 unsupervised network layers, the BAM's error decreased rapidly but then fluctuated around a high MSE value, reflecting results from the previous section. However, with 10 or 13 unsupervised network layers, the BAM's error lowered and stabilized at MSE values of ≈ 0.02 and < 0.001, respectively.

Fig. 3.18(b) shows average decision zones for all conditions. For $l = 1$, the model produced a linear decision boundary, misclassifying the task. For $l = 4$, the general solution from the previous section was observed again. For $l = 7$, the overlapping region (peak of the parabola) was seen to lose its smoothness and showed a shift in the classification behaviour. This change was accentuated for $l \geq 10$, where "pockets" appeared around the three exceptions, showing the appropriate solution for the task.

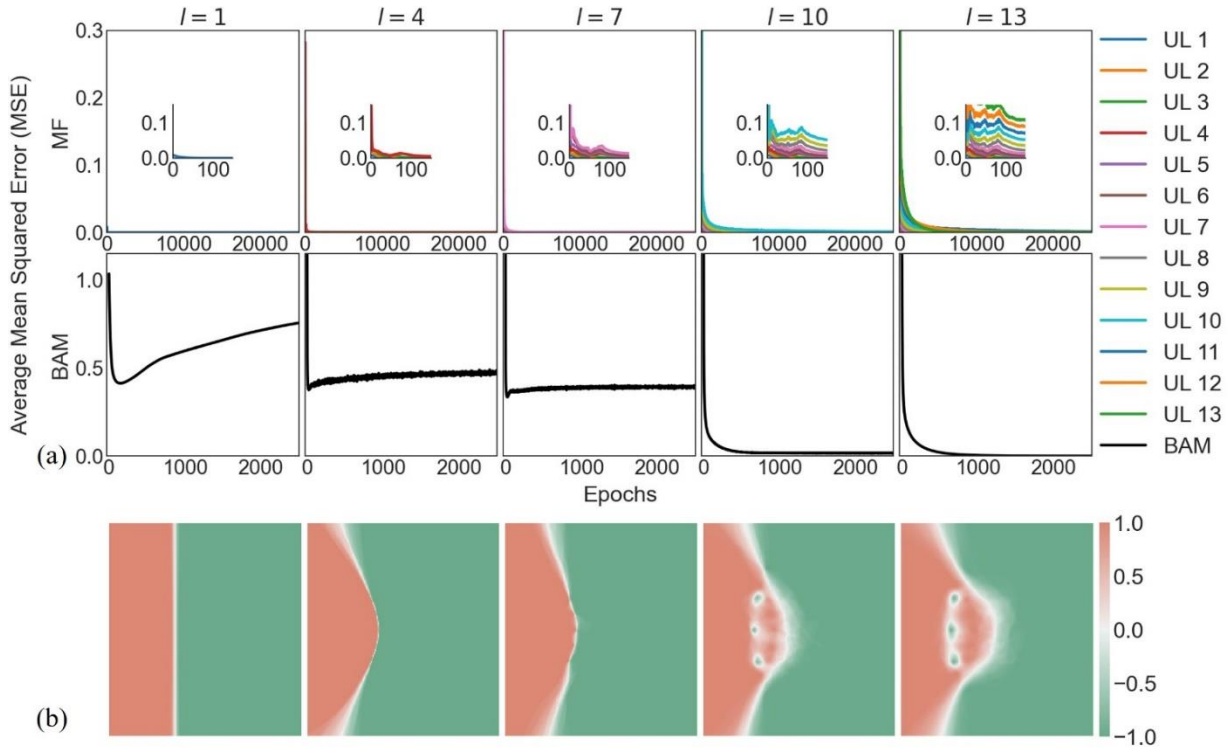

Fig. 3.18. (a) Average mean squared error for the Suns task under different number of unsupervised network layer (*l*) conditions. b) Decision boundaries for the Suns task under the same conditions.

Table 3.3 shows the average recall performance and the standard deviation for each condition of the Suns task. In general, as *l* increased, the recall performance increased. Notably, a perfect performance (100% input-target association with a standard deviation of 0) was achieved with *l* = 13.

Table 3.3. Average recall performance and standard deviation for the Suns task.

		Number of unsupervised network layers (*l*)				
		1	4	7	10	13
Suns	Mean	80.00	87.45	87.85	98.70	100.00
	Std.	0.00	3.10	3.32	3.05	0.00

3. 5. 3. Impact of the Number of Unsupervised Network Layers Discussion

In section 3.5, results showed that by varying *l* while fixing *u* at 500, the model could get closer to providing 100% performance consistently. This perfect input-target associative behaviour came with $l = 13$ (Table 3. 3), where the SL achieved an MSE value near zero (Fig. 3.18(a)), and the three previously problematic exceptions were separated adequately in the decision boundaries (Fig. 3.18(b)). This indicates that increasing the depth of the MF impacts the final generated representations making it possible to learn to classify the Suns task. As shown in Fig. 3.18(b) for conditions $l \geq 10$, decision boundaries exhibited pocket-like zones around the three conflicting points. In other words, the MF-BAM learned exceptions to the general rule. However, it is hypothesized that the more inputs learned in those contradictory positions, the more these pockets would become pronounced, and the fewer units and layers would be needed to learn the task. Further testing should study this frequency effect. If true, this could give important insights into how learning exceptions to a general rule may depend on the architectural structure and not just exposure, as seen in young children learning to speak and call all animals, either cats or dogs.

Interestingly, in all the simulations, a common trend emerged in the MF during learning. As the first layer learned and its MSE tended toward zero, the second layer's error increased for a short time before decreasing back toward zero (Fig. 3.18(a)). This was repeated for all the subsequent layers until the last layer's error stabilized near zero. In other words, as a layer learned, its generated representations kept fluctuating, and once stabilized, low errors would be reached. This allowed the subsequent layer to receive consistent information and eventually stabilize. Moreover, the patterns of each layer can be extracted and used separately, meaning different behaviours could be observed even if the inputs do not change. For example, patterns from the 4th

layer could be used by an SL to capture the general form. At the same time, a different SL could simultaneously learn patterns from the 13th layer to capture the exceptions to the general rule.

In summary, results from section 3.5 showed how the MF-BAM could be tuned to either overlearn or produce a more generalized solution. This behaviour change came from manipulating the *l* parameter of the MF component. Interestingly, depending on the depth of the MF, the generated patterns could be used to train the BAM module to produce a linear (generated patterns from the shallow layers), nonlinear (generated patterns from the deeper layers), or an overfitted response (generated patterns from the deepest layers). In other words, with a MF of various layers, it is possible to have all three behaviours simply by extracting the appropriate generated pattern from either the shallower, deeper, or deepest layers.

So far, sections 3.4 and 3.5 have shown that the MF-BAM can learn both a general and specific solution to a nonlinear task. These behaviours were obtainable by manipulating the *u* or *l* architectural parameter in isolation. As results have suggested so far, these parameters influence the nature of the generated patterns produced by the MF. Depending on this nature, the BAM module had poor or good performances. Thus, the goal of the next section was to understand better what type of architecture (e.g., shallower with more units vs. deeper with fewer) was more favorable. As such, both parameters *u* and *l* were systematically manipulated to study their impact on the model's behaviour. Furthermore, learning time in the MF module was varied to evaluate if any architecture brought better performances in a time restraint setting.

3. 6. Impact of Manipulating the Number of Units (*u*) and Unsupervised Network Layers (*l*)

In this section, both the number of units per layer (*u*) and number of unsupervised network layers (*l*) were manipulated. The goal was to measure the effect of varying these architectural parameters on the 2-bit parity problem and determine how it affected the BAM module's learning and associative performances. Furthermore, the MF's max learning time (*e*) was constrained and varied to further determine its effect on the BAM module. Finally, to better understand how *u* and *l* could impact the learning speed of the supervised module, a follow-up analysis was performed on the generated outputs from the different architectures.

3. 6. 1. Methodology

The 2-bit task was used again in this simulation. As such, the inputs and targets were generated in the same fashion as described in section 3.4.1, i.e., inputs contained 2^2 bipolar vectors of length 2, associated with targets of either 1 or -1. To evaluate the effects of *u*, *l*, and *e* in the MF component, different architecture conditions were created. Thus, *u* varied from 5 to 100, *l* from 1 to 5 (for comparison, $l = 10$ was also investigated), and *e* was constrained from 10 and 2500 for the MF only. Furthermore, the BAM module was allowed to learn for 2500 epochs for all conditions. To help determine if the BAM had learned the 2-bit task correctly, the number of epochs required to obtain a MSE < 0.0001 in the BAM module was recorded for each trial. Additionally, average performance was computed over 100 repetitions per condition to establish reliability. Both learning and recall procedures followed what was described in section 3.3. Finally, density plots were generated to better illustrate the impact of *u* and *l* combinations on the BAM. These plots examined the values of each node of the generated patterns in each condition analyzed.

3. 6. 2. Results

Fig. 3.19(a) shows the MF-BAM's average recall performance on the 2-bit task according to each architecture condition and time constraints in the MF. In general, results show that the outcome on performance when changing *u* and *l* varied for different time constraints. With little learning times ($e = 100$), performance was higher for shallower (smaller *l*) and smaller networks ($u \leq 30$) rather than deeper ($l > 1$) or larger networks ($u \geq 70$). For longer learning times ($e \geq 500$), the opposite effect could be seen, i.e., higher *l* and *u* led to better performances. Alternatively, when $e = 250$, a mixed effect was observed, as 2 to 5 ULs yielded better performances than 1 or 10.

More specifically, results showed that if learning time was too low ($e = 10$), no configuration of the MF's parameters could lead to a consistent BAM performance increase. However, when $e = 100$, the performance achieved from ULs 1 to 4 resembled an inverted "U" shape. That is, it initially increased at low levels of *u*, reached a maximum performance before 30 units, and subsequently plummeted towards zero as *u* further increased. In general, in this time constraint, the lower *l* was, the better performance was. For instance, the highest "peak" performance achieved was 98% when $l = 1$ (dark green) and $u = 23$. In fact, for $l = 5$ (light green) or 10 (yellow), performance did not increase.

When $e = 250$, performance when $l = 1$ still exhibited the same inverted "U" shape but with slightly higher scores, where 100% was achieved once with $u = 25$. For $l = 2$ (orange), performances improved to 91% at $u = 10$, fluctuated between 80% to 90% until $u = 80$, and slowly decreased to 70% at $u = 100$. For $l = 3$ (purple), performances fluctuated between 80% and 90% from 10 to 100 units, while for $l = 4$ (red) or $l = 5$, performances similarly fluctuated between 90% and 99% from 10 to 100 units. Interestingly, performances when $l = 10$ (or 1) also resembled an inverted-U curve. Under the $e = 500$ constraint, performances fluctuated between 97% and 100%

for the $l = 10$ and $u \geq 20$, $l = 5$ and $u \geq 35$, or $l = 3$ and $u \geq 45$ architectural conditions. For $l = 3$, performances fluctuated between 90% and 99% when $u \geq 10$.

Steady 100% performance was achieved when $e = 1000$ for the $l = 10$ and $u \geq 25$, $l = 5$ and $u \geq 50$, $l = 4$ and $u \geq 65$, and $l = 4$ and $u \geq 80$ conditions. Like before, no significant performance changes were observable with $l = 1$ or $l = 2$. Finally, when learning was completed at $e = 2500$, consistent 100% performance was achieved when $l \geq 3$ and $u \geq 15$. Additionally, their performance curves for the $l = 1$ and $\underline{l} = 2$ conditions remained similar to those found in the $e = 500$ and $e = 1000$ constraints. Thus, showing marginal changes as learning time increased passed 500 epochs.

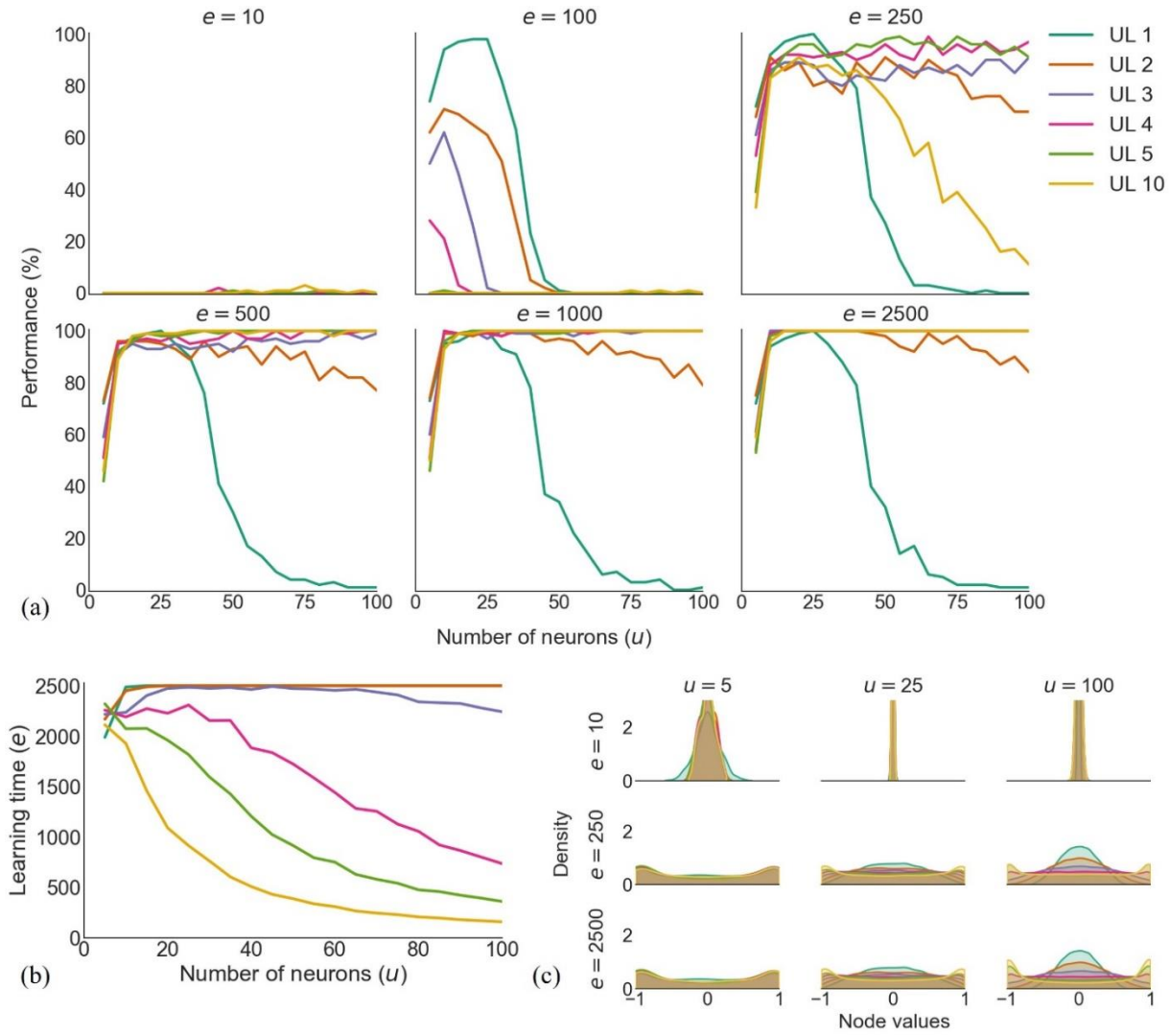

Fig. 3.19. (a) 2-bit average performance for different architectures and learning time conditions. (b) BAM module's number of epochs (*e*) to reach MSE < 0.0001 in function of the number of units and unsupervised network layers in the MF. (c) Density plots of the generated patterns from the MF in function of the number of units and unsupervised network layers conditions and time constraints.

Fig. 3.19(b) shows the average learning time for the BAM to stabilize to an MSE < 0.0001 according to the number of units and layers in the MF. If the number of epochs reached the maximum of 2500, it indicated that the network was unable to fully learn the task. As results showed, for $l \geq 3$, an increase in u and l lowered the number of epochs required for the BAM to achieve an MSE of less than 0.0001. This was evident when the comparison was made between l

= 4 under the u = 5, u = 50, and u = 100 conditions. A decrease in the number of epochs was observed (2257, 1722, and 357, respectively). Increasing l to 10 led to a further reduction of average learning time, where for u = 5, u = 50, and u = 100, the epochs went from 2111 to 385, then to 155, respectively. On the flip side, for $l \leq 2$, the average learning time was faster for lower u values and slower for higher ones. This was seen when l = 1, as it took on average 1987 epochs for u = 5 and 2486 epochs for u = 10, while an MSE error smaller than 0.0001 was not achievable within 2500 epochs for $u \geq 15$.

Fig. 3.19(c) shows the density plots of the distribution of values in each unit of the generated representations at different e, u, and l constraints. Generally, two distributions could be seen: normal (bell-shaped) or bimodal. On average, representations created from networks with low l resulted in a more normal distribution with a density peak centered at 0. In contrast, those from networks with higher l had a bimodal distribution with density peaks at the bipolar extremes -1 and 1. Exceptionally, if e = 10, the distribution was normal with a very narrow and high-density peak around zero regardless of u and l, showing that 10 epochs were insufficient for the MF to generate patterns that were not linked to the values of the weight initialization. In general, results showed that an incremental increase of l produced a gradual transition from a normal distribution to a bimodal distribution. This can be seen for e = 2500 and u = 100, as the normal distribution (for l = 1) gradually flattened and became bimodal at -1 and 1 as l increased. On the other hand, increasing u values did not change the distribution under a given l but led to a stronger density peak whether normal or bimodal. Similar results were observed when e was increased.

3. 6. 3. Impact of Manipulating the Number of Units (*u*) and Unsupervised Network Layers (*l*) Discussion

Results in section 3.6 have shown the impact of manipulating *e, u,* and *l* on the BAM module. Generally, results showed that when the MF's learning was not too restricted ($e \geq 500$), increasing *u* and *l* resulted in better performances on the 2-bit task. This also allowed the BAM module to exhibit faster and steadier learning, i.e., shorter learning times and perfect performances throughout the 100 trials.

Of course, if insufficient learning times (e =10) were given, the network was not able to perform the task regardless of the architecture. If more time was allowed (e ≥100), then better performances were observed. In those cases, it seems that the initial layer (UL1) was finally stabilizing, as shown in Fig. 3.19(a), which was independent of the *u* constraint. Additionally, results showed that in short learning times (e = 100), having a single layer was more advantageous than having multiple ones. Again, stabilization of the UL1 was essential if any subsequent layers were to provide an adequate representation to the BAM module. Suppose enough time is allowed (e = 250), then deeper layers ($2 \leq l \leq 5$) can fully deploy their power, as shown by the better performances of the BAM module. This suggests that if the weights of the deeper layers have enough time to distance themselves from their initial value, performance in the BAM module will increase. This also means that generating a viable solution does not require the MF layers to stabilize fully. Still, to achieve perfect learning of the input-target associations for every trial, enough time should be given and enough layers, as increasing *l* seemed to raise the likelihood of generating sets of linearly separable patterns. However, increasing only the depth is not always enough. This is especially true for fewer units per layer ($u \leq 15$). Therefore, the role of *u* may be to create a big enough feature space to generate a steady solution if the number of layers is high

enough. In other words, both *u* and *l* must be considered together to guarantee perfect performance all the time. An interesting example of this is highlighted when $u = 25$ and $l = 1$, as it resulted in 100% performance.

A parallel trend was echoed in the density plots (Fig. 3.19(c)), as deeper and wider architectures seemed to be linked with better performances and faster learning times. These results emphasized that the architectural parameters of the MF were indeed changing the "nature" of the generated patterns. Bimodal distribution resulted in faster learning times for the BAM module as well. However, this type of distribution did not fully guarantee success in solving the task. This was seen in the $u = 5$ and $e = 2500$ constraints, as despite being bimodal for all $l > 1$, no consistent perfect performances were observed. Further investigations are needed to understand better how these patterns change over time and why these shapes impact the transformation from nonlinearity to linearity required for the BAM.

If learning a task by the BAM must be done quickly, then more time should be allocated to a MF with more layers and units. Although learning in the MF for these conditions was longer, this was compensated for by fast learning times in the BAM module (as seen in Fig. 3.19(b)). Thus, increasing both *u* and *l* parameters was very beneficial. However, suppose the total number of units must be considered. In that case, results from section 6 suggest that 100 units should be organized throughout more layers with fewer units than having all of them in one layer.

To sum up, both *u* and *l* in the MF contribute towards generating a linearly separable representation. Which parameter is the most important? It is hard to give a definitive answer. It will depend on the restriction applied; fewer layers with fewer units outperforms its counterpart (when the MF is restricted to 100 epochs and BAM's learning time is disregarded). Still, in other paradigms, the opposite becomes true (when total learning time is to be minimized and

performances maximized). However, results have shown that increasing *u* and *l* do not impair performance. Thus, if resources are limitless and the best performances are to be sought, having a deeper and larger model will only require a bit more learning time for the MF component as a counterweight. This situation may be closer to cognition, as in the brain, billions of neurons organized in parallel systems are used, and in most cases, learning continues throughout its life span.

So far, it has been shown in the previous sections that different nonlinear behaviours can be obtained by modifying the architecture of the MF-BAM. However, it remains to be investigated if the MF can enable continuous online learning without needing a reset in weights values between tasks. Moreover, will the generated representation be self-adaptable if novel inputs for a given task are used? Thus, in the next section, the MF-BAM performance was studied under the condition of learning multiple tasks one after the other.

3. 7. Continuous Online Learning

Humans are often said to be flexible, able to shift from learning one task to another seemingly effortlessly. However, this is not always the case for ANNs, as they can stay stuck to a previous solution when learning a different task. This refers to sequentially learning various tasks by presenting the input-target pairs incrementally and without resetting the model's weight values between tasks. Thus, it is a means to show some plasticity during learning. To evaluate this, a sequence of tasks, namely the 2-bit, Moons, and Eclipse, were presented in succession to the MF-BAM, and its behaviours (decision boundaries) were recorded over time.

3. 7. 1. Methodology

The 2-bit, Moons, and Eclipse tasks were created as described in section 3.4. Each task was sequentially presented to the MF-BAM in the following order: 2-bit, Moons, and Eclipse. The architecture parameters were set to $u = 500$ and $l = 4$, following results in section 3.4, where this architecture was shown to solve all three tasks successfully. The model had 500 epochs to learn a task before introducing the next one. Likewise, this number was based on results from section 3.6, which showed that this number of epochs was enough. Decision boundaries were produced after each epoch to assess the model's behaviour across time. The learning and recall procedure from section 3.3 was followed. For consistency, 100 trials were conducted, and the average learning error and decision zones were reported.

3. 7. 2. Results

Average learning errors (MSE) across time (e) after the introduction of all three tasks for both the MF and BAM are shown in Fig. 3.20(a). Results indicated that in the BAM, there was an increase in error (MSE values ≥ 1) followed by a gradual decrease towards 0 after a new task was introduced during learning. In each task, error achieved low values (MSE ≤ 0.0001) within the 500 epochs the task was presented. Likewise, an increase in error was observed for all ULs of the MF once each task was presented. However, the extent to which error increased was smaller with the introduction of each successive task.

In Fig. 3.20(b), decision boundaries in function of the BAM's learning (e) are shown. Notably, decision boundaries were seen to transition from one task to another successfully, and 100% recall performance was achieved. This was gradually done within the time window of each task being presented. As the 2-bit task was introduced ($e = 1$), the decision boundary first exhibited random values until forming the correct zones. Once the second task was introduced ($e = 501$), this shape

moved toward the appropriate general structure for the Moons. Similarly, after the Eclipse task was introduced (e = 1001), the decision boundary slowly transitioned toward the correct general solution. In each case, the final decision boundaries were like the previous results in section 3.4 (Figs. 3.12(b) and 3.17). Furthermore, a proper decision boundary was usually achieved within the first 400 epochs of a presented task.

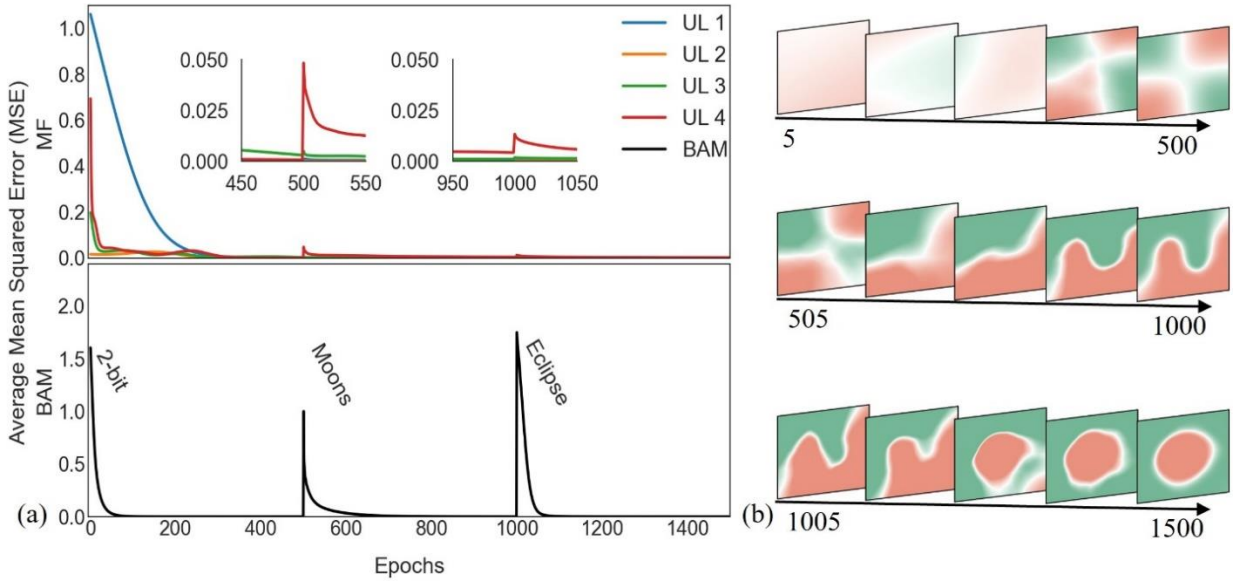

Fig. 3.20. (a) Average mean squared error for the continuous learning task. (b) Decision boundaries of the continuous learning task.

3. 7. 3. Continuous Online Learning Discussion

In summary, results have shown that the MF-BAM can sequentially learn multiple nonlinear tasks. This was expected since an essential feature of associative memories is that they are plastic (i.e., updating to new data; (Zilly et al., 2021)). This usually results in the model "forgetting" old tasks since weights stabilize to the new solution, to the detriment of performance on the previous task. This was seen in the BAM module, as learning errors always spiked over 1 MSE when learning a new task. As such, the BAM would steadily adapt to the new task and forget old ones. This behaviour was also shown in the decision boundaries, seen in Fig. 3.20(b).

However, this was different for the MF module. After introducing a new task to the MF, learning errors across unsupervised network layers would increase to a lesser extent. This might indicate that generated hidden representations were not drastically modified when learning a new task. In other words, some features generated from task A could be maintained when learning task B. In turn, the MF could be interpreted as accumulating knowledge from new tasks. In other words, random stimuli would be sufficient for the MF to learn to organize itself and help distinguish the totality of the input space. This may be akin to how the early stages of the sensory system mature as the brain is exposed to new stimuli (Ackman & Crair, 2014). As such, continuously training the MF could eventually lead it to generate patterns that can be reused in the BAM to relearn previous tasks or even future ones successfully.

Although the model has been shown as a promising approach to dealing with nonlinear associations, all previous tasks contained only two classes. In many instances, a learner will be faced with a multiclass problem. Therefore, the following section investigated whether the MF-BAM could learn nonlinear tasks while maintaining previously observed behaviours in a multiclass setting.

3.8. Multiclass Nonlinear Task

This section assessed the MF-BAM using the Spiral task, a 2D continuous valued nonlinear task consisting of 3 classes. When faced with an *n*-class problem, multiple methods can reduce a multiclass task into different binary-class problems to facilitate learning. However, the goal for this section was to avoid these methods and see if the network could learn a nonlinear multiclass task without any preprocessing and through simple input-target associations. Using the Spiral task was the next logical step, as results from a 3-class setting can be extended to tasks with even more

classes. Lastly, the task was kept in a 2D setting to allow the visualization of the network's behaviour through its decision boundaries.

3. 8. 1. Methodology

The 3-class Spiral task was created similarly to previous 2D tasks, where 2-dimensional input vectors ranging with values from -1 to 1 were generated. The target vectors were changed from 1-dimensional to 3-dimensional ones to ensure that behaviour would also scale to higher-dimensional target patterns. As such, the first class was represented by the target vector [1, -1, -1], the second by [-1, 1, -1], and the third by [-1, -1, 1]. Each class contained 40 inputs, giving 120 data points for the entire dataset. Fig. 3.21(a) shows the visual representation of the 3-class Spiral task. Learning and recall followed the procedure in section 3. However, predictions for decision boundaries were determined as follows: if the output was the correct class, it was considered a success. Otherwise, it was considered a failure. The simulation was repeated 100 times, and average decision zones were reported. For this simulation, the architecture was set to u = 500 and l =10, following results from sections 4 and 5. The number of epochs (e) was set to 25000 for the MF for investigating stability purposes and at e = 2500 for the BAM.

3. 8. 2. Results

Fig. 3.21(b) shows the average learning error for both the MF and BAM components. In both cases, learning followed the same trend from previous results: Higher MSE that decreased over time. In addition, it also showed how from 0 to 50 epochs in the MF module, slight increases of MSE were gradually seen in each UL after previous layers stabilized. Moreover, as previously observed, the deeper the UL, the higher the learning error spiked for that layer. Despite these peaks, the MSE from all layers steadily decreased toward 0. For the BAM component, MSE values

decreased rapidly within 247 epochs. The model's average classification behaviour can be seen in Fig. 3.21(c). In all 100 trials, the MF-BAM could learn the Spiral task perfectly. Decision zones confirm that the classification behaviours of the MF-BAM were stable. Additionally, decision zones showed how each class was well separated by an uncertain area where no correct classification could be observed.

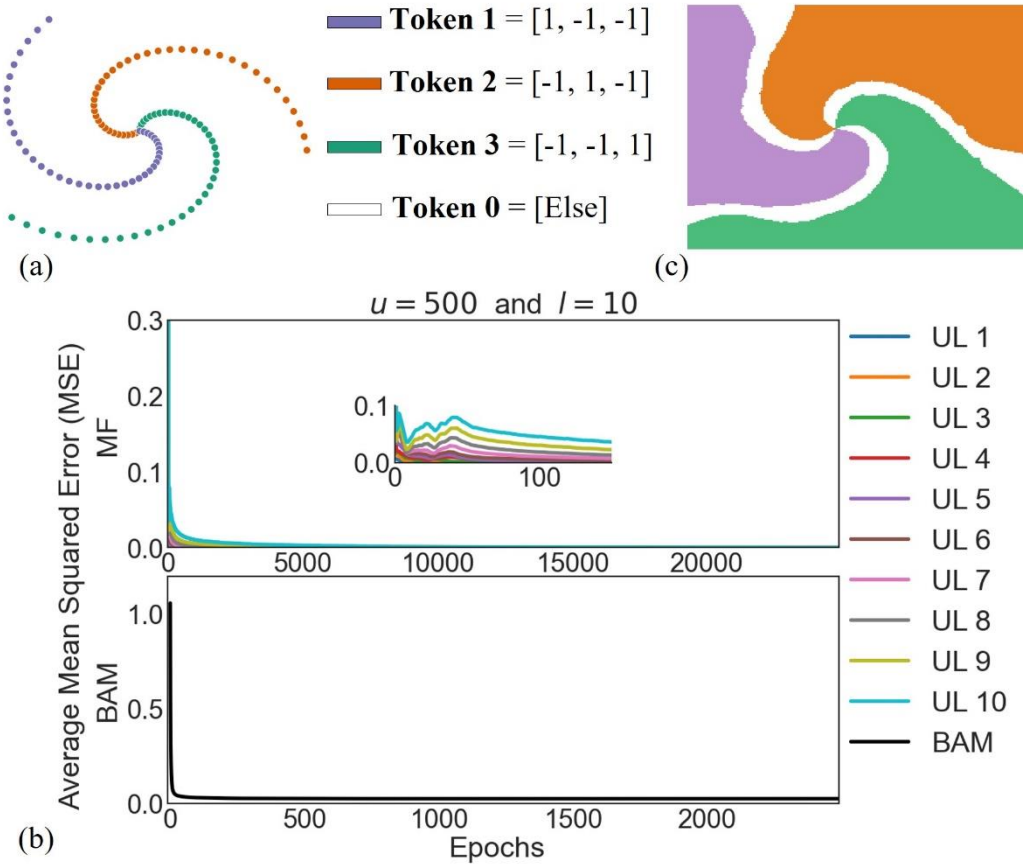

Fig. 3.21. (a) 3-class Spiral task. (b) Average mean squared error for the MF and BAM components on the 3-class Spiral task. (c) Decision boundary for the Spiral task.

3. 8. 3. Multiclass Nonlinear Task Discussion

This simulation evaluated whether the MF-BAM could deal with more than two classes in a nonlinear task. Results showed that the model could classify the three classes perfectly. Going from 2-class to 3-class did not have an impact on the general behaviours of the model. Furthermore,

learning in the MF and BAM stabilized to a consistent solution. These findings suggest that the general trends found in previous sections also apply to an *n*-class setting.

3.9 Chapter III Discussion

This chapter proposed the Multi-Feature extraction bidirectional associative memory - Bidirectional Associative Memory (MF-BAM), a RNAM composed of multiple unsupervised network layers and a single supervised network layer. This model could solve various bipolar and continuous values nonlinear classification tasks within the neurodynamic perspective.

Of importance, each unsupervised network layer interacts with only local information without backpropagation of the error. In addition, the same transmission and Hebbian learning were used throughout the network. Thus, it maintained the same internal dynamics for all its layers. Changes in behaviour only came from manipulating the architectural parameters of the MF. This significantly increases biological and cognitive plausibility (O'Reilly, 1998; Thomas & McClelland, 2008). Furthermore, the MF-BAM did not rely on preprocessing nor required external intervention (e.g., early stopping). Table 3.4 shows a summary of comparisons between the modified BAM (Chartier & Boukadoum, 2006a), the bidirectional backpropagation BAM (Adigun & Kosko, 2019; Kosko, 2021), the combined Multi-Layered Perceptron-BAM (Wu et al., 2018), and the MF-BAM.

Table 3.4. Different properties from cognitively inspired models.

	BAM	Backprop-BAM	MLP-BAM	MF-BAM
Biological realism	Yes	No	No	Yes
Distributed representation	Yes	Yes	Yes	Yes
Error-driven	Yes	Yes	Yes	Yes
Hebbian learning	Yes	No	No	Yes
Activation Spreading	Yes	Yes	Yes	Yes

Bidirectionality	Yes	Yes	No	Yes
Learns Nonlinear tasks	No	Yes	Yes	Yes

This chapter's main takeaway is that the MF-BAM can effectively learn a wide range of nonlinear associations. This can be explained by revisiting the MF-BAM's two modules: the Multi-Feature extraction (MF), composed of unsupervised network layers (ULs), and the BAM, which is a supervised network layer (SL). Given that the latter remained untouched from previous research, success on nonlinear tasks depended solely on the MF. Consequently, for the BAM to learn a given nonlinear task, the MF had to generate representations from the inputs that shared a linear relationship with the corresponding targets. Results showed that this was achieved steadily, as the model showed 100% performance across all nonlinear tasks if enough units per layer (u) or unsupervised network layers (l) were used.

In addition to these findings, this chapter revealed several other interesting properties of the MF-BAM model. Most notably, the shift from a linear to a nonlinear decision boundary was not an all-or-nothing process. Instead, it was a smooth transition controlled by changes in architecture. Given enough layers, increasing the number of units would gradually develop a stable solution (Section 3.4 and 3.5). Adding layers can make the model highly specific (overfitted) to a task. In other words, it can learn exceptions to rules. Given the dynamic taking place during learning, where layers incrementally stabilize one at a time, the generated patterns from the first few layers can be used quickly. This means it would be possible to sacrifice performance to tackle problems needing faster response times by using shallower layers. For better performances, increasing learning times and using deeper layers would suffice. This tailored answer goes by the speed-accuracy trade-off literature in ANNs (Huang et al., 2017; Li et al., 2019). There is also a possibility of using the output from each hidden layer rather than only the final one. Each layer

could connect to a single BAM to obtain diverse behaviours (linear, nonlinear, specific) from the same inputs. These structural and dynamic properties found in the MF-BAM align with what is observed in humans. For example, individuals can give different answers when encountering the same stimulus, which can change as more exposure time is provided or context (Hesselmann et al., 2008; Standage et al., 2014).

Section 3.6 presented a comprehensive analysis of the impact of both *u* and *l* parameters which yielded significant findings. Specifically, deeper configurations were found to have a clear advantage over having more units onto a single layer. The distribution of values in the generated patterns was also identified as essential, with bimodal distributions proving more advantageous than normal ones. Furthermore, the generated patterns from the deeper and larger MF architectures not only shared a linear relationship with their corresponding targets but were also much more distinct. Each UL (FEBAM) may be doing a feature extraction process akin to something like principal component analysis (PCA) by generating a pattern that has low-level statistical features that represent the input pattern's intrinsic information (Chartier et al., 2007). Repeating this process over multiple layers would mean that each deeper generated pattern becomes a more salient representation of the distinguishing features of the input pattern. In other words, it decorates the input patterns (Rolon-Merette et al., 2018). This could explain why the BAM could learn quickly after the stabilization of the MF. These findings may provide valuable insights into why the brain's neural networks are recursively organized and why they are thought to function in an associative manner (Buckner & DiNicola, 2019; Wang & Cui, 2018).

Furthermore, it was shown that the generated patterns in the MF would remain relatively consistent even with the addition of new inputs. This was indicated by results from section 7, where the increase in MSE in the MF was smaller and smaller as new tasks and more inputs were

introduced. This suggests that some information was being conserved and that the model was not entirely subjected to catastrophic forgetting (Kemker et al., 2018). This hints towards an advantage of presenting more inputs to the MF. Consequently, it is imaginable to train an MF on the entire input space through random exposure. Although this process would require more time, once trained, the MF could be reused in an online setting without catastrophic forgetting if the dimensionality of the input does not increase. Furthermore, results from section 7 suggested that as the MF learns, the computational cost (number of epochs) of learning a new task diminishes. It would be interesting to verify this property's extent, as it is crucial in modelling how humans learn online, as information is constantly being captured, processed, and stored throughout their lifespan.

Despite these interesting properties, some limitations must be addressed in future studies. This research maintained an equal number of units in each layer of the MF. This was done to isolate the effects of having more layers or units. Since each layer can have an independent number of units, further research could explore utilizing various layer sizes and how this impacts behaviour. Previous studies have shown that in a single FEBAM, it is possible to adapt the number of units to generate diverse behaviours, such as creating prototypes (compression) or exemplars (expansion) (Giguère et al., 2007; T. Rolon-Mérette et al., 2019). As such, it would be possible to use the bidirectional nature of the model to generate prototypical categories based on small u values and which layer depth information is compressed on. It may also be possible to incrementally add units and layers to a model without retraining the entire model, similar to other models such as the adaptive resonance theory (ART) model (Carpenter & Grossberg, 1988). This would be highly advantageous from an engineering perspective since it could automate finding an optimal configuration for a given task. However, this may be less crucial when modelling human cognition, as the architecture is typically considered fully developed, much like an adult brain, and

does not change significantly depending on the task. Similarly, it was shown throughout the paper that even if the number of units or layers is greater than necessary, it does not hinder performance. This suggests that setting the architecture is not a trial-and-error process, and using more units and layers can be the default approach.

Finally, the bulk of this chapter focused on classical nonlinear tasks using two-dimensional input vectors. This was necessary to examine and visualize the model's task performance. However, the BAM is not optimal for two-dimensional input and may have slower learning times. In addition, higher dimensional input is more relevant to human cognition. Results from the *N*-bit, where $N \geq 3$, have shown that the MF-BAM can scale up to learn tasks in higher dimensions. Similarly, it was shown that the MF-BAM could deal with a nonlinear task containing three classes. Future studies could investigate if changing the representation of a nonlinear task to a higher dimension can impact behaviour and if there is a relation between the dimensionality of the input and the generated patterns on the number of learned classes.

3.10. Chapter III concluding remarks

This chapter has shown that the MF-BAM can successfully learn nonlinear tasks when given sufficient units and layers. The representations generated from the last unsupervised network layer of the MF component are linearly separable, ensuring the BAM's success. The model achieves this without adding external factors and relying on architectural changes alone. The internal mechanisms remain homogeneous, utilizing the same local learning and transmission function while maintaining the general bidirectional structure.

This work underscores the capability of a multilayered recurrent neural associative memory to handle nonlinear tasks, though it lacks a mechanism for controlling information flow within the

network. This leads to a reliance on the final layer for training and utilization in every instance, which is suboptimal for several reasons. Primarily, the training of the final layer can be time-intensive, as evidenced by a 13-layer MF requiring 25,000 epochs to final a representation that facilitates the BAM's learning of the Suns task. Additionally, the ultimate representation produced by the final layer may not always be necessary or desired. For example, in the Suns task, while the 13th layer generated a highly nuanced decision zone encompassing all exceptions, a simpler and more generalized separation, potentially offered by the 4th layer, could suffice. Yet, the current model precludes the exclusive use of the representation from the 4th layer.

The issue is inherent to the MF-BAM model's use of bipolar encoding, which does not accommodate a true absence of information. For instance, in previous chapters, a white pixel is encoded as -1, which does not represent the true absence of information. This complicates or even prevents operations such as masking or inhibition. Thus, the external introduction of a zero vector by the experimenter would be necessary to accurately mask or inhibit information. It would be more efficient and plausible if the network could self-generate a zero vector in response to a specific context or task requirements. To do so, a bipolar transmission function with stable attractors at 1 and 0 would be necessary. However, bipolar encoding is used in BAMs for its superior learning efficacy and behavioral outcomes, meaning there is a lack of suitable binary transmission functions. Consequently, the forthcoming chapter will present a novel transmission function that retains the learning strengths of bipolar encoding while integrating additional cognitive features.

4. Chapter IV

Are Zeros and Ones All You Need In A Recurrent Neural Associative Memory

Abstract

Recurrent Neural Associative Memories (RNAMs) are a branch of Artificial Neural Networks (ANNs) inspired by brain neurodynamics and are valued for cognitive modelling. The architecture of these models is typically restricted to one network, which limits the complexity of tasks they can learn. This may be due to their transmission function, as this function must have two stable fixed points to support learning attractors. These functions traditionally use bipolar encoding for learning speed and performance. However, bipolar encoding cannot represent a true absence of information or exhibit inhibition behaviour. This limits the proper control of information flow in a multi-network architecture to external intervention. Additionally, there is always the inherent storage of the complementary attractor for every learned stimulus which act as spurious attractors. Conversely, binary encoding is more reflective of biological systems but usually yields suboptimal learning. As such, this study introduces a new transmission function for binary encoding with fixed points at 0 and 1. Through application and analysis in a Bidirectional Associative Memory (BAM) model, it is shown that the new transmission function preserves the learning advantages of bipolar encoding and significantly reduces spurious attractors, thereby improving recall accuracy in noise. It is also shown that various new properties are obtained, such as inhibition and short-term memory, allowing multi-network architectures. This advancement not only brings RNAMs closer to biological realism but also sets the stage for learning more complex cognitive tasks.

Keywords: *Binary encoding, Transmission Function, Associative Learning, Cognition*

4. 1. Introduction

Associative learning and associative memory are important processes in human cognition that have been studied extensively in a wide array of fields (Kumar et al., 2015; Lansner, 2009; Letzkus et al., 2015; Mayes et al., 2007; Puig et al., 2014; Suzuki, 2008). Research has shown that they emerge from the parallel and distributed computations of the brain's vast network of interconnected neurons (Christophel et al., 2017; Goldstein, 2015). However, the precise mechanisms through which these networks encode and process environmental stimuli to produce associative behaviour still needs to be understood. Artificial Neural Networks (ANNs) offer a platform for simulated and repeatable exploration of these mechanisms in a non-invasive setting to help explain these complex neural processes (Eliasmith, 2013; Hassabis et al., 2017; Mareschal et al., 2000; Yang & Wang, 2020). Recurrent Neural Associative Memories (RNAMs) are a subclass of ANNs designed to model associative memory. They share similarities with typical ANNs, consisting of units (artificial neurons) and weighted connections. They have recurrent, or feedback, connections similar to those found in Recurrent Neural Networks (RNNs). However, RNAMs are often more concerned with modelling the neurodynamic processes underlying cognition than the application-based objectives of most RNNs. As such, RNAMs focus on the network dynamics representing the brain's neuronal state space as attractors (Amit & Amit, 1989). In contrast to ANNS, who typically use the backpropagation learning algorithm to update their weights, RNAMs usually use Hebbian learning to store stimuli as attractors, which adheres to Hebb's classic postulate (1949) that "neurons that fire together wire together" (Lim, 2021). Even though different learning rules have been applied to RNAMs, the Hebbian rule remains amongst the most sought-after due to its biological plausibility.

Thus, learning in RNAMs is driven by co-occurrences in the environment. In other words, weights are incrementally adjusted to form attractors according to local error and not global information. In addition to Hebbian learning, RNAMs aim to be constrained by other biologically plausible mechanisms like biological realism, distributed representations, error-driven learning, and bidirectional activation propagation (O'Reilly, 1998; Pulvermüller et al., 2021). Despite this, RNAMs have shown many interesting properties. For instance, RNAMs manage noisy stimuli effectively, as their recurrent characteristics help guide the distorted stimuli toward the attractor that corresponds to the original stored versions. Such an iterative noise filtering mechanism relies upon the stabilization properties associated with multiple attractors and, consequently, the successful matching of inputs to attractors through the learning process. As such, the learning dynamics of RNAMs are influenced by the quantity and characteristics of their stable states. Specifically, when a greater number of correlated patterns are incorporated, these networks tend to produce nonlinear combinations of these patterns. This process can lead to the emergence of spurious, i.e. garbage, attractors, in addition to the intended inputs (Chartier & Proulx, 2005).

Beyond the Hebbian learning rule, the transmission function is a central element of RNAMs, critically influencing the number and character of attractors generated during the learning process. Consequently, a nuanced understanding of its cognitive implications and role in attractor formation is imperative. In RNAMs, a transmission function with two stable fixed points is desired for proper weight convergence (Storkey & Valabregue, 1999). The way stimuli are encoded can also wield significant influence. Most RNAMs prefer stimuli encoded in a bipolar (-1,1) format over a binary (0,1) one. In classical RNAMs, like the Hopfield network or some Bidirectional Associative Memory (BAM), bipolar encoding is essential to guarantee learning and achieving distinctive input representations (Kosko, 1988). In more recent BAMs which use a converging

learning rule, bipolar encoding is preferred as the informative value of -1 aids in the convergence of weights, enabling the model to learn faster (Acevedo-Mosqueda et al., 2013; Adigun & Kosko, 2019; Kosko, 2021; Li et al., 2021; Morissette & Chartier, 2013; Shi & Zeng, 2020). However, binary encoding and having the zero as a natural attractor could allow the zero vector to be used in an inhibition mechanism, which could lead to the network performing different mechanisms and operations that could only be implemented manually in a bipolar setting. For instance, to inhibition is difficult to achieve in bipolar encoding, as inhibiting a bipolar pattern requires its exact opposite to be used, which would require prior knowledge of exactly each pattern to inhibit and very problematic when inputs are not totally bipolar, i.e. might have some values between -1 and 1. With binary, the absorbent property of the zero can be used to shut down any signal. As such, it would be better to implement mechanisms such as gating and masking which are important part of multi-network architectures, such as ensemble networks, in controlling the flow of information (Ganaie et al., 2022; Sagi & Rokach, 2018). Such multi-network dynamics have been key in recent advances of ANNs and, coincidentally, are present in our brain's biological neural networks and believed to be an important part of associative memory (Abbott, 2006; Grewe et al., 2017; Spooner et al., 2020).

Bipolar encoding poses other challenges. For instance, the ability to generate orthogonal stimuli with bipolar encoding is more complex. For decades, it’s been common knowledge that learning orthogonal stimuli is much easier than correlated ones (Personnaz et al., 1986). By using a representation that capitalizes on the value of 0 rather than -1, generating these types of stimuli could become simpler. Additionally, having only stable attractors at -1 and 1 does not facilitate a model representing sparsity. Remedying this could help improve learning performances quite drastically in RNAMs (Liu et al., 2021; Zhang et al., 2023). Importantly, it would adhere to the

strong evidence that sparseness is found in the brain, thus increasing the model's biological plausibility (Ahmed et al., 2020; Beyeler et al., 2019; Földiak, 1990) and potentially increasing learning performance (Hoefler et al., 2021). Also, encoding 0s and 1s in a binary representation can efficiently represent streams of spikes. This alignment with actual neural mechanisms could provide valuable insights into the brain's inner workings and form a bridge to spiking neural networks (Brendel et al., 2020; He et al., 2019). From a cognitive plausibility perspective, binary encoding would be preferred to bipolar encoding.

Despite these drawbacks, bipolar encoding remains prevalent in most RNAMs, partly because it facilitates faster learning speeds and due to the absence of suitable binary transmission functions. Therefore, this study aimed to create a binary transmission function to emulate the appealing properties of bipolar encoding while circumventing its limitations, potentially enabling the modelling of more complex cognitive mechanisms and the integration of more biologically plausible representations.

4. 1. 1. Background

RNAMs have a long history and can be traced back to the 70s with the brain-state-in-box (BSB). The BSB was an important landmark as it could perform auto-association by storing orthogonal patterns in fixed points where noisy patterns could be reconstructed by the recurrent interaction between the weight matrix, the inputs, and their feedback loop (Anderson, 1983; Anderson et al., 1977), However, RNAM gained popularity in the 80s due to the Hopfield network (Hopfield, 1982). This model showed how a simple one-shot Hebbian learning rule could be used to perform auto-association despite being in the presence of correlated stimuli. The Hopfield network also relied on the recurrent interaction between the weight matrix, the inputs, and their feedback loop to push its signals toward attractors. It introduced nonlinearity to the model with a binary (0, 1)

step function (transmission/activation function) and encoding was shifted to bipolar during learning and back to binary for recall. This strategy was short-lived, as the network was limited and struggled to learn highly correlated stimuli. This was because strict Hebbian learning inputs interact in an additive fashion thus impairing the model's learning capability. Also, using different encodings for learning and recall also posed a justification problem. The solution to this problem was to use the bipolar (-1, 1) representation consistently (Hopfield, 1982).

This step function was replaced with the logistic function, which gained in popularity due to its use in feedforwards ANNs like the Multi-layer Perceptron to increase performance and since it is easily derived (Basheer & Hajmeer, 2000; Jain et al., 1996). Interestingly, this logistic function comes from the solution to the classic Verhulst differential equation, which was proposed to model population growth. A simplified general Verhulst logistic function can be expressed as:

$$(4.1)\quad \frac{dx}{dt} = \gamma(1 - x)x$$

where γ is a general parameter (constant). Fig 4.1 illustrates this quadratic function for $\gamma = 1$. As shown, the function has 2 fixed points but only x = 1 (black circle) is stable while x = 0 (white circle) is unstable (repeller). This posed an important problem from a neurodynamic perspective due to this single stable fixed point. In fact, the use of a transmission function, like the logistic, in an RNAM architecture, hinders its ability to adequately use its recursive properties to represent the brain's neuronal state space as attractors.

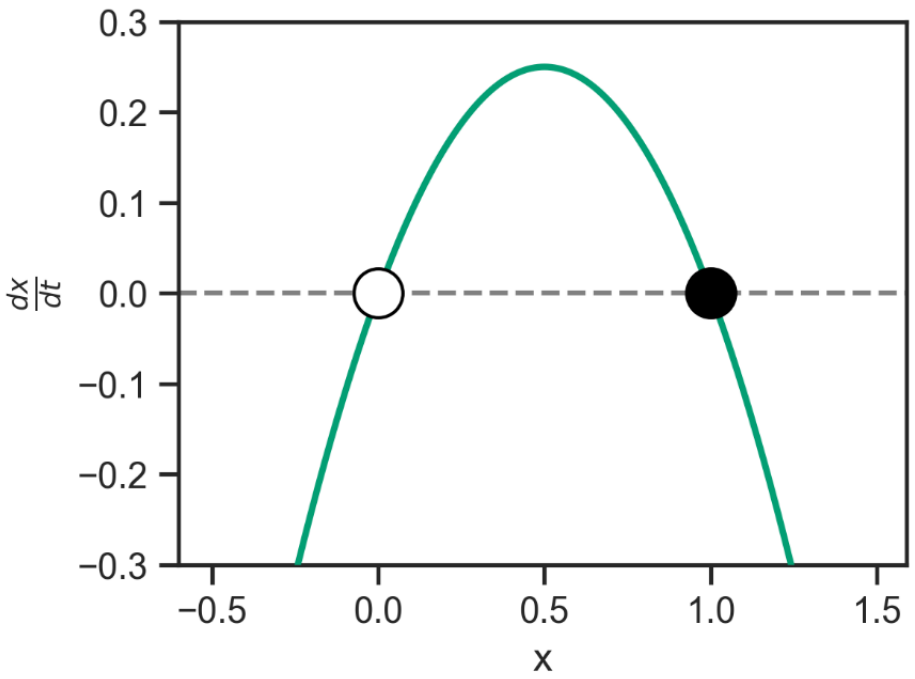

Fig. 4.1. The Verhulst logistic function when γ = 1. The white dot shown an unstable fixed point at x = 0 and the black dot a stable fixed point at x = 1.

To address this issue, Chartier and Boukadoum (2006a) proposed a bipolar transmission function in conjunction with the use of Hebbian and anti-Hebbian learning in a bidirectional associative memory (BAM) like model (the bidirectional hetero associative memory; BHM). Interestingly, their transmission function was adapted from the classic Verhulst equation previously described by extending it to a cubic form:

$$(4.2)\ \frac{dx}{dt} = x - x^3$$

Where x represents the input value. Of interest, as shown in Figure 4.2, this cubic function provides two stable fixed points at the values of -1 and 1 and an unstable point at the value of 0.

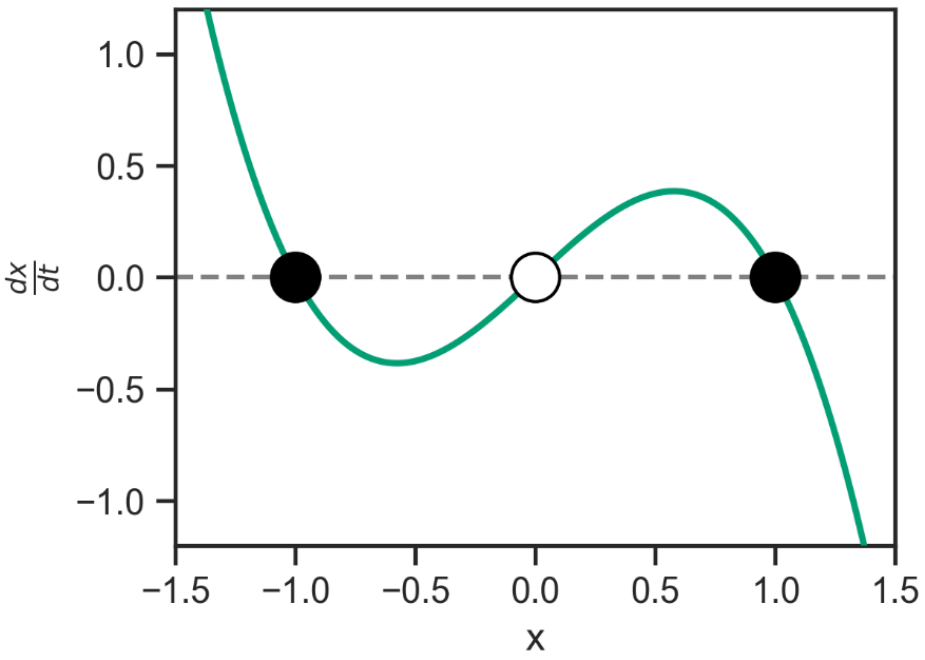

Fig. 4.2. The fixed points in the bipolar transmission function, a white dot shows an unstable fixed point at x = 0 and the two black dots show stable points at x = -1 and x = 1.

To ensure network scalability, hard limits at -1 and 1 were introduced as can be seen in Equation 4.3 using Euler approximation:

$$(4.3)\ \mathbf{o}_{i[t+1]} = \begin{cases} 1, if\ \mathbf{a}_{i[t]} > 1 \\ -1, if\ \mathbf{a}_{i[t]} < -1 \\ (\delta + 1)\mathbf{a}_{i[t]} - \delta\mathbf{a}_{i[t]}^{3}, Else \end{cases}$$

Where δ is a general transmission parameter, i is the index unit, t the time index, $\mathbf{a}_{i[t]}$ represent the activation and $\mathbf{o}_{i[t+1]}$ the output. The shape of this transmission function can be seen in Figure 4.3.

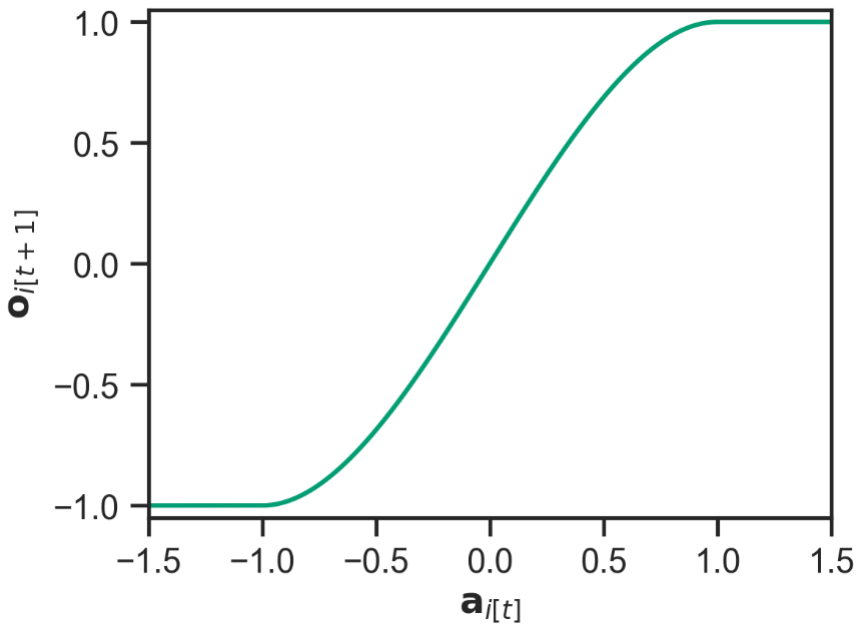

Fig. 4.3. The bipolar transmission function with hard limits when δ = 0.5.

4. 1. 2. Problematic

However, this model remains bound by the drawbacks of bipolar representations. Hence, the challenge undertaken by this research is to develop a BAM that retains the RNAM properties and advantages found in bipolar encoding while incorporating a more cognitive plausible representations and creating new properties to unlock multi-network architectures. As such, this paper aims to introduce a novel binary transmission function that when combined with Hebbian and anti-Hebbian learning can store correlated binary patterns. For comparison purposes, both the binary and bipolar transmission functions will be implemented in a BAM model. Learning times, the number of spurious attractors and noise tolerance will be evaluated. Then, additional properties possible with the new transmission function within multi-network architectures. As such, this study aims to contribute to the ongoing efforts of developing ANNs capable of learning complex

cognitive tasks while adhering to principles grounded in the brain's cognitive and biological processes.

This paper is divided as follows. In section 4.2, the novel binary transmission function is presented and analyzed. To evaluate the learning properties and performance within a neurodynamic framework, we utilized a supervised and unsupervised version of a Bidirectional Associative Memory (BAM) model, which is introduced in Section 4.3. In Section 4.4, the learning performance was measured in various association tasks. Different measures were used such as learning speed, attractor properties and recall performance to compare binary and bipolar encoding. Then, additional properties gained with binary encoding are explored. Specifically, one-shot learning, gating and masking are implemented in sections 4.5, 4.6 and 4.7, respectfully. Section 4.8 presents a detailed discussion of the results and their implications, and a concise conclusion is provided in Section 4.9.

4. 2. Analysis of the binary transmission function

In this section, an alternative approach to bipolar encoding that has stable fixed points at 0 and 1 is proposed. This binary transmission function was designed in part to ensure that the symmetry generated by its bipolar counterpart was broken. As such, it should eliminate the complement spurious attractor being generated. A general binary ordinary function that displays this desired property can be expressed as:

$$(3.4)\ \frac{dx}{dt} = \gamma(3x^2 - 2x^3 - x)$$

To better understand the number and nature of the attractors produced by Equation 3.4, the sign of the roots of the function must be looked at. As shown in Figure 3.4, when γ =1, three intercepts

exist at 0, 0.5, and 1. The values of 0 and 1 have a negative slope and are stable attractors. The value of 0.5 has a positive slope and is thereby unstable.

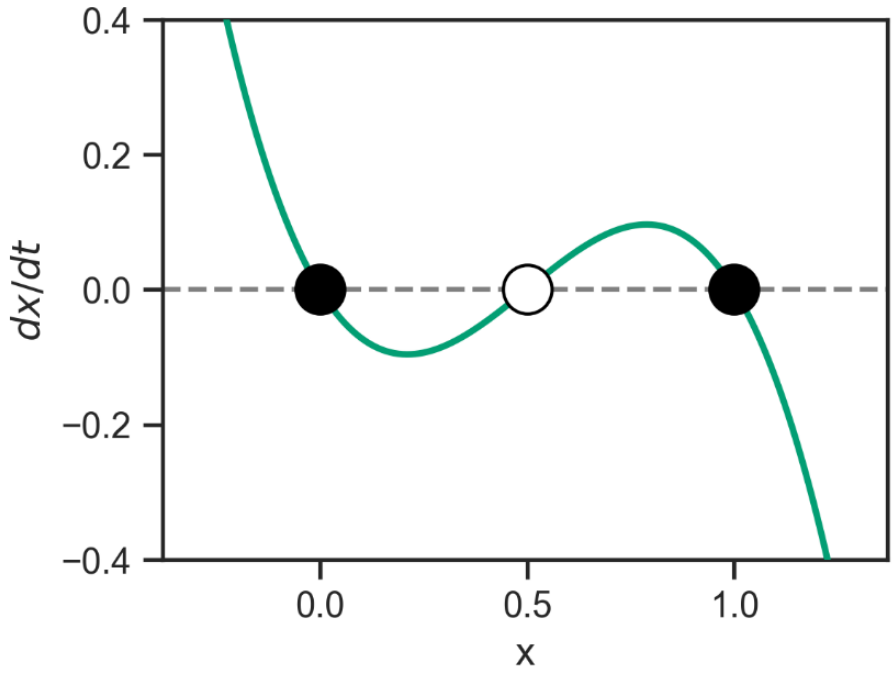

Fig. 4.4. Depicts the general binary ordinary function at γ = 1, revealing two stable fixed-points and one unstable fixed-point.

Interestingly, when solving Equation 4.4, two solutions are obtainable. If γ =1, these are:

$$(4.5a)\ x_{[t]} = \frac{1}{2} - \frac{\sqrt{e^t}}{2\sqrt{4e^c + e^t}}$$

$$(4.5b)\ x_{[t]} = \frac{1}{2} + \frac{\sqrt{e^t}}{2\sqrt{4e^c + e^t}}$$

In other words, if a value is below the unstable point, Equation 4.5a should be employed. In the opposite case, Equation 4.5b should be used. Figure 4.5 shows both solutions for different values of *c*. This further confirms that as time progresses, the values of $x_{[t]}$ will converge to either stable attractor of 0 or 1 depending on if $x_{[0]}$ is lower or higher than 0.5, respectfully.

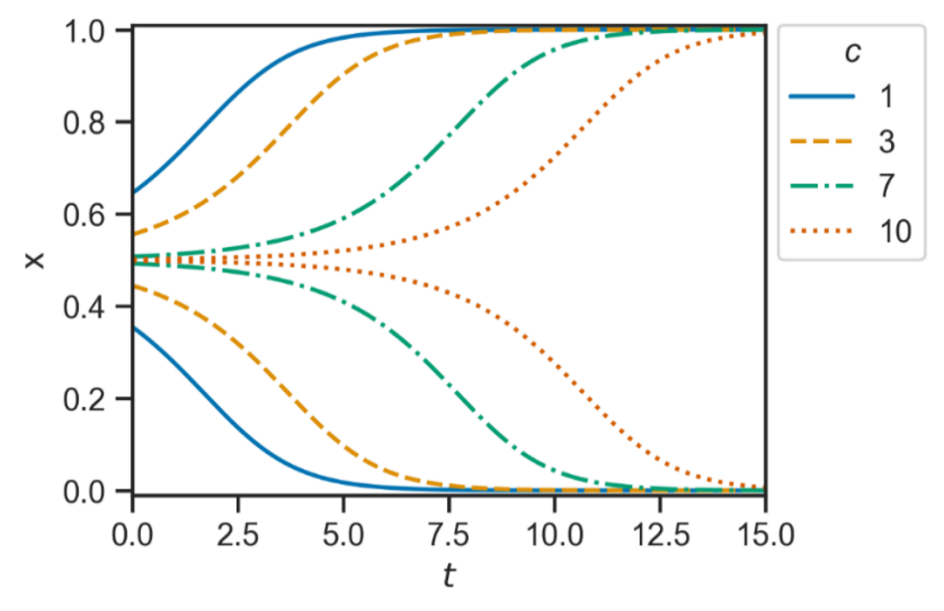

Fig. 4.5. The two solutions of the binary transmission function according to different values of *c*.

By integrating Equation 4.4, we obtain the energy function. This function shows that the rate of change (energy) decreases and is the lowest at the attractor basins. In other words, it is guaranteed that any noisy patterns (higher energy) will end in one of the attractors. The binary's energy function for a one-dimensional system is described by Equation 4.6 and can be seen in Figure 4.6:

$$(4.6)\quad \mathrm{E(x)} = \gamma\left(\frac{\mathrm{x}^4}{2} - \mathrm{x}^3 + \frac{\mathrm{x}^2}{2}\right)$$

As it can be observed, the energy landscape resembles a "W" with minima being reached at the stable attractors of 0 and 1 and the central peak at the unstable one of 0.5.

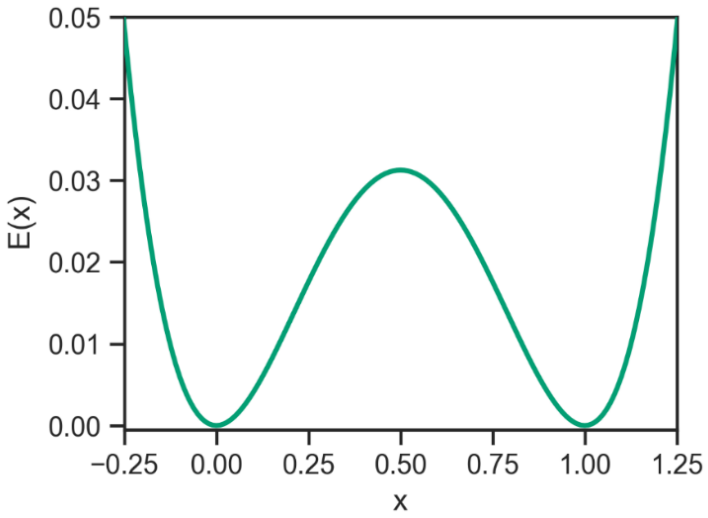

Fig. 4.6. The Energy landscape in a 1-dimensional system (1 node).

The energy function can also be generalized to an *N*-dimensional setting, as described by Equation 4.7:

$$(4.7)\ \mathrm{E}(\mathbf{x}) = \gamma\left(\frac{\mathbf{x}^{\mathrm{T}}(\mathbf{W}\mathbf{x})^{3}}{2} - \mathbf{x}^{\mathrm{T}}(\mathbf{W}\mathbf{x})^{2} + \frac{\mathrm{x}^{\mathrm{T}}(\mathbf{W}\mathbf{x})}{2}\right)$$

This permitted a visualization of the energy for a two-dimensional system, $\mathbf{x} = [x_1, x_2]$, and confirmed the scalability of the transmission function. As shown in Figure 7a) and 7b), the energy is minimized at the binary corners of each dimension.

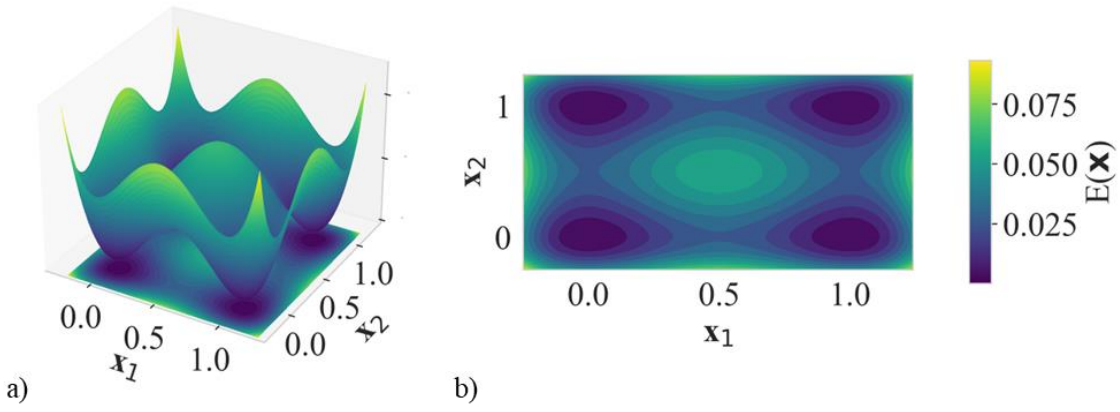

Fig. 4.7. The Energy landscape in a 2-dimensional system is illustrated in a) using a three-dimensional plot and b) projected onto a two-dimensional plane.

In sum, E(x) decreases along trajectories, and the state vector globally converges towards lower energy regardless of the dimensionality. Equilibrium occurs at the basins of the vector field where local minima correspond to stable attractors and local maxima correspond to unstable ones. To implement the function in an RNAM, a Euler approximation was used. This was done by converting the continuous-time Equation 4.4 into discrete-time, transforming it into Equation 4.8:

$$(4.8)\ x_{[t+1]} = \delta\gamma\left(3x_{[t]}^2 - 2x_{[t]}^3 - x_{[t]}\right) + x_{[t]}$$

And, if $\gamma = 1$ then we can set $\phi = \delta\gamma$ and Equation 4.8 can be further simplified as:

$$(4.9)\ x_{[t+1]} = \phi\left(3x_{[t]}^2 - 2x_{[t]}^3 - x_{[t]}\right) + x_{[t]}$$

Figure 4.8 showcases a Euler approximation ($\phi = 0.1$) and confirms that Equation 4.9's outputs can converge towards two stable fixed points over time.

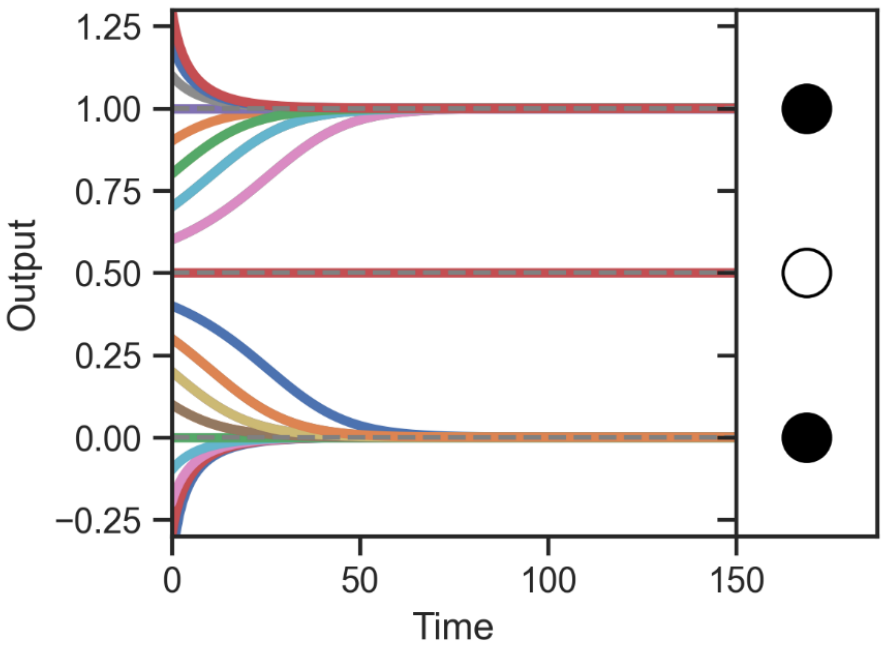

Fig. 4.8. The binary transmission function's outputs converging towards these fixed points from different initial conditions (colored lines) following a Euler approximation ($\phi = 0.1$).

To investigate the behaviour around each attractor, i.e., fixed point, cycle, or chaotic attractor, and to determine which values of ϕ should be used to ensure the creation of fixed points, an analysis using the Lyapunov exponent (λ) was needed. This enabled to quantify the rate at which the network's trajectories converge or diverge. To perform this analysis, the derivative of Equation (4.9) was first required and is described in Equation 4.10.

$$(4.10)\ \frac{dx_{[t+1]}}{dx_{[t]}} = \phi\left(6x_{[t]} - 6x_{[t]}^2 - 1\right) + 1$$

With the derivative in hand, the Lyapunov exponent could be approximated, and the bifurcation diagram could be examined to assess chaotic behaviour. When looking at the bifurcation diagram (Figure 4.9a), a steady state can be obtained if $0 < \phi < 2$. Moreover, when observing the Lyapunov exponent in Figure 4.9b), it shows that any value satisfying $0 < \phi \leq 1$ will lead to a monotonic convergence to the fixed points. While when using $1 < \phi \leq 2$, it will lead to an alternate convergence. Thus, when the value is higher than 2, a period-doubling bifurcation to chaos ($\lambda >$ l) is observed. As such, any value that satisfies $0 < \frac{d\mathbf{x}_{[t+1]}}{d\mathbf{x}_{[t]}} \leq 1$ for ϕ will suffice. Consequently, to simplify Equation 4.10, the value of $\phi = 1$ was chosen. This simplifies the Equation while also filling all the requirements to make any input converge to a stable fixed point.

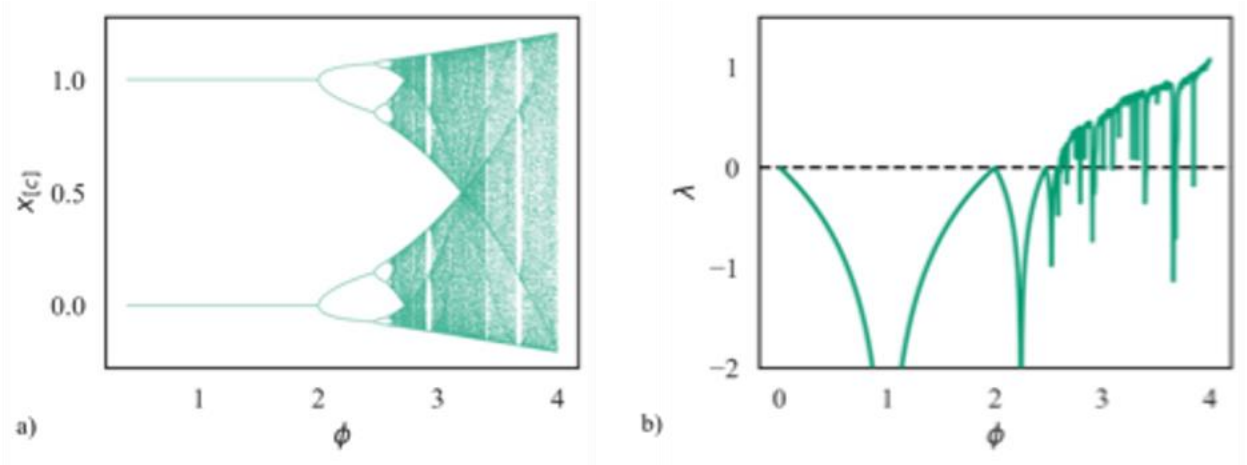

Fig. 4.9. a) The bifurcation diagram and b) an approximation of the Lyapunov exponent.

Equation 4.9 can now be simplified into Equation 4.11, which encapsulates the summed-up behaviours:

$$(4.11).\ x_{[t+1]} = 3x_{[t]}^2 - 2x_{[t]}^3$$

The final step before implementing the function in a neurodynamic model resided in accounting for the neural unit's absolute maximum and minimum values. This was done by adding hard limits to values below 0 or above 1. Thus, the final form of the binary transmission function takes the expression given by Equation 4.12:

$$(4.12).\ \mathbf{o}_{i[t+1]} = \begin{cases} 1, if\ \mathbf{a}_{i[t]} > 1 \\ 0, if\ \mathbf{a}_{i[t]} < 0 \\ 3\mathbf{a}_{[t]}^2 - 2\mathbf{a}_{[t]}^3, Else \end{cases}$$

Where i is the index unit, t the time index, $\mathbf{a}_{i[t]}$ represents the general activations and $\mathbf{o}_{i[t+1]}$ the outputs. Thus, the novel binary transmission function with two stable fixed-point attractors at the values of 0 and 1 and an unstable fix point at the value of 0.5 can be seen in Figure 4.10.

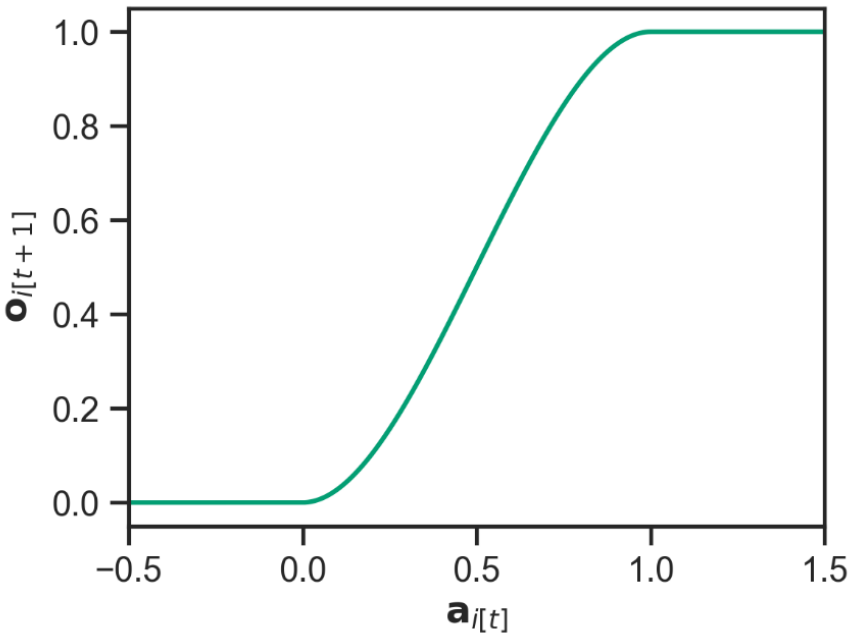

Fig. 4.10. The binary transmission function with hard limits at 0 and 1.

4. 3. Model

To examine the differences between binary and bipolar transmission functions from a neurodynamic standpoint, a BAM model was used. Introduced by Kosko (1988), BAMs are one of the most popular classes of RNAM, as they can perform auto- and hetero-associations in a bidirectional fashion by storing their learned information in different attractors, see (Acevedo-Mosqueda et al., 2013) for a comprehensive review on BAMs. This bidirectionality gives these models great power when faced with noisy versions of their learnt patterns. From the vast repertoire of BAMs, a modified version, called the BHM (Chartier & Boukadoum, 2006a) was employed. This model has shown numerous characteristics when using bipolar encoding, including the ability to learn auto and hetero-associations, handle correlated and grey-level stimuli, perform online learning, exhibit noise tolerance, and engage in sequence learning, among other features. Additionally, this model combines Hebbian and anti-Hebbian learning, meaning weight updates are done entirely with local information. The presence of anti-Hebbian learning was crucial towards using the binary representation, as it provided the negative feedback needed to counterbalance the strict positive values in the inputs (Storkey & Valabregue, 1999). Furthermore, an unsupervised version of this BAM, named the FEBAM, was created with a slight change to its architecture by removing a set of external connections (Chartier et al., 2007). The learning rule and other functions remained the same. As such, both supervised and unsupervised versions will be used and will be described in this section.

4.3.1 Architecture

The architecture of this BAM version is illustrated in Figure 4.11a. As it can be seen, it consists of two layers, each receiving a pair of the association between the pair of inputs $\mathbf{x}_{[0]}$ and $\mathbf{y}_{[0]}$, where

the weight matrices **W** and **V** serve as feedback connections between them. The number of units in each layer (dimensionality) is given by *M* and *N*.

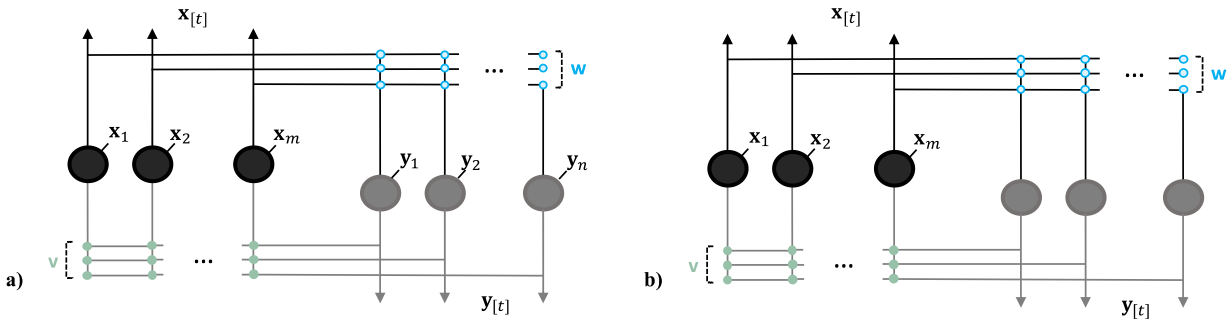

Fig. 4.11. The architecture of the (a) Bidirectional Associative Memory (BAM) and (b) The Feature-Extracting BAM (FEBAM).

The FEBAM's architecture is shown in Fig. 4.11b, where the only difference is the absence of the second set of external inputs ($\mathbf{y}_{[0]}$). Instead, representations, or features $\mathbf{y}_{[0]}$, are created though the cycle in the network as described in (Chartier et al., 2007).

4.3.2 Transmission function

The activations in both networks are obtained by Equations 4.13a and 4.13b:

$$\text{(4.13a)} \quad \mathbf{a}_{[c]} = \mathbf{W}\mathbf{x}_{[c]}$$

$$\text{(4.13b)} \quad \mathbf{b}_{[c]} = \mathbf{V}\mathbf{y}_{[c]}$$

From Equations 4.13a and 4.13b, the binary transmission function can be fully expressed by Equations 4.14a and 4.14b:

$$\text{(4.14a)} \quad \forall i, \ldots, N, \mathbf{y}_{i[c+1]} = f(\mathbf{a}_{i[c]}) = \begin{cases} 1, if\ \mathbf{a}_{i[c]} > 1 \\ 0, if\ \mathbf{a}_{i[c]} < 0 \\ 3\mathbf{a}_{[c]}^2 - 2\mathbf{a}_{[c]}^3, Else \end{cases}$$

$$(4.14b)\quad \forall i, \dots, M, \mathbf{x}_{i[c+1]} = f\left(\mathbf{b}_{i[c]}\right) = \begin{cases} 1, if\ \mathbf{b}_{i[c]} > 1 \\ 0, if\ \mathbf{b}_{i[c]} < 0 \\ 3\mathbf{b}_{[c]}^{2} - 2\mathbf{b}_{[c]}^{3}, Else \end{cases}$$

Where *M* and *N* are the number of units in each layer, *i* is the index unit and *c* the iteration cycle.

4.3.3 Learning rule

In both BAM and FEBAM, weights are updated according to a Hebbian/anti-Hebbian learning rule, defined by Equations 4.15a and 4.15b:

$$(4.15a).\quad \mathbf{W}_{[k+1]} = \mathbf{W}_{[k]} + \eta\left(\mathbf{y}_{[0]} - \mathbf{y}_{[c]}\right)\left(\mathbf{x}_{[0]} + \mathbf{x}_{[c]}\right)^{\mathrm{T}}$$

$$(4.15b).\quad \mathbf{V}_{[k+1]} = \mathbf{V}_{[k]} + \eta\left(\mathbf{x}_{[0]} - \mathbf{x}_{[c]}\right)\left(\mathbf{y}_{[0]} + \mathbf{y}_{[c]}\right)^{\mathrm{T}}$$

Here, $\mathbf{x}_{[0]}$ and $\mathbf{y}_{[0]}$ are the initial inputs (or $\mathbf{y}_{[0]}$ are the initial generated representations in the FEBAMs case), $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$ are their respective reconstructed patterns, η is the learning parameter, *k* is a given learning trial and **W** and **V** are the weight matrices who converge when $\mathbf{x}_{[0]} = \mathbf{x}_{[c]}$ or $\mathbf{y}_{[0]} = \mathbf{y}_{[c]}$. During learning in all simulations, *c* was set to 1.

4.3.4 General learning procedure

Learning is conducted according to the following procedure:

1. Randomly select a pattern from the list of inputs to obtain the initial inputs $\mathbf{x}_{[0]}$ and $\mathbf{y}_{[0]}$.
2. Cycle through the network using the output functions (Equations 4.14a and 4.14b) to obtain the outputs $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$. This process is outlined in the FEBAM in (Rolon-Merette et al., 2019).
3. Update the weights according to the learning rule (Equations 4.15a and 4.15b).
4. Repeat steps 2 to 4 until the minimum mean squared error between the initial patterns, $\mathbf{x}_{[0]}$ or $\mathbf{y}_{[0]}$, and outputs, $\mathbf{x}_{[c]}$ or $\mathbf{y}_{[c]}$, is achieved.

4.3.5 General recall procedure

Similarly, recall is performed according to the following procedure:

1. Select an initial input from the list, denoted as $x_{[0]}$.
2. Compute $\mathbf{y}_{[0]}$, $\mathbf{x}_{[c]}$ and $\mathbf{y}_{[c]}$ for c cycles or until convergence ($\mathbf{x}_{[c]} = \mathbf{x}_{[c+1]}$ or $\mathbf{y}_{[c]} = \mathbf{y}_{[c+1]}$).
3. Repeat steps 1 to 2 for each initial stimulus.

4. 4. Simulation I – Neurodynamic learning

In this section, multidimensional association tasks were used in the BAM and FEBAM to compare learning performance with new binary transmission function to the bipolar transmission function. First, a simple auto-association task comprising three-dimensional data was used to visually compare learning behaviour in the new binary transmission function to the bipolar transmission function. Then, to gain a thorough comparison, a Montecarlo simulation was conducted to examine learning in higher dimensional auto-association tasks in both supervised (BAM) and unsupervised (FEBAM) models. The task involved learning varying numbers of randomly generated 100-dimensional correlated input vectors. These vectors consisted of 0s and 1s for binary encoding or -1s and 1s for bipolar encoding.

4. 4. 1. Procedure

The first task involved learning the following three-dimensional input vectors: (1,1,1), (1, -1, -1) and (1, -1,1) for bipolar encoding and (1,1,1), (1,0,0) and (1,0,1) for binary. This task was used to be able to visualize attractors in a three-dimensional setting. After learning was completed, random vectors in the 3-dimensional space were generated and used in recall, following procedure 4.3.5. This was performed to examine the presence of attractors.

In the Monte Carlo simulation, 100-dimensional vectors were randomly generated. The number of randomly generated inputs was varied from 1 to 50 to create memory loads from 1 to 50%. Three metrics were measured to compare learning performance between binary and bipolar encoding: the learning time, the percentage of spurious recall and noise tolerance using a pixel flip recall task. The learning time was measured using the number of epochs until a minimum means square error (MSE) of 10^{-6} between the stored and actual input vectors was achieved or when a maximum of 1000 epochs was reached. After learning was completed, 10,000 random vectors were used for recall to measure the proportion of random vectors that stabilized in a spurious state. Then, a pixel flip recall task was used to measure the network's ability to filter noise from a noisy input by flipping random input pixels, i.e. changing 0s to 1s in binary encoding, or vice versa, and -1s to 1s in bipolar encoding, or vice versa. The number of flipped pixels ranging from 0% to 50%. An example of various levels of pixel flipping can be seen in Figure 4.12 with a 100-dimensional vector representing the letter “O” shown in a 10 by 10 grid. For each encoding type and memory load, 100 trials were conducted for learning, spurious attractor recall, and recall under noise. Each trial used new inputs, randomly generated recall inputs to measure spurious attractors, and randomly generated pixel flip noise.

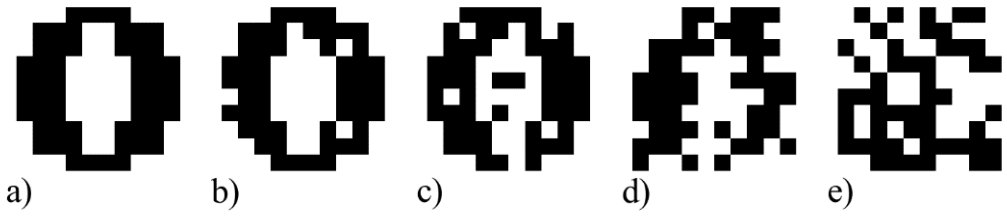

Fig. 4.12. Example of noisy inputs with a) the original input, b) 5 pixels flipped (5%), c) 10 pixels flipped (10%), d) 25 pixels flipped (25%), and 50 pixels flipped (50%). Here, white pixels signify a 0 in binary or -1 in bipolar notation, and black pixels represent 1.

In each task, learning in the FEBAM and BAM followed procedure 4.3.4 and stopped once 1000 epochs were reached or a minimum error of 10^{-6}. the learning parameter (η) was set to 0.01. Recall followed procedure 4.3.5 until $\mathbf{x}_{[c]} = \mathbf{x}_{[c+1]}$ or for a maximum of 100 recall cycles Finally, δ was assigned a value of 0.5 in the bipolar transmission function. Furthermore, across both networks and all tasks, the same parameters were set. In both the FEBAM and BAM models, weights were randomly initialized within the range of -0.1 to 0.1, using a seed to guarantee identical initial conditions for both networks. It is to note that the BAM's weights could be initialized to zero, but results did not significantly differ, so they were omitted.

4. 4. 2. Results

In the three-dimensional task, attractors are shown in Figure 4.13. As shown in Figure 4.13a), in the bipolar encoded BAM, inputs converged on six attractors; these were the values of the input vectors (1,1,1), (1, -1, -1) and (1, -1,1) and their complement (spurious attractors). The same attractors are present in the bipolar FEBAM (Fig. 4.13b). In the binary BAM, there were only four attractors, one for each input vector and one at (0,0,0), as shown in Figure 14c). The same four attractors are present in the FEBAM (Fig. 4.14d) but in a lower proportion of zero vectors. Consequently, the binary transmission function eliminated the complementary attractors, leading to a decrease in the number of spurious attractors, with the zero vector remaining as the sole spurious attractor.

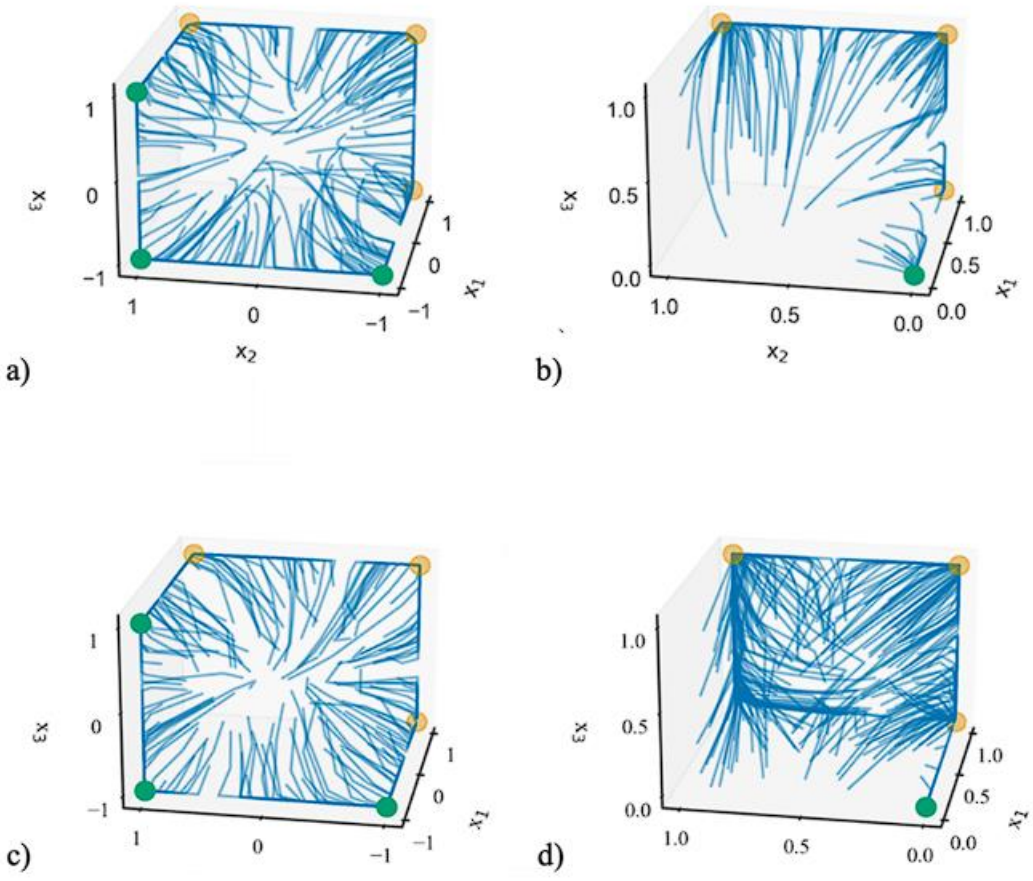

Fig. 4.13. Attractors formed when learning three 3-dimensional patterns in a) the modified Bidirectional Associative Memory (BAM) using bipolar encoding or b) using binary encoding with the proposed transmission function and the FEBAM using c) bipolar or d) binary encoding. Yellow dots represent attractors corresponding to inputs, green dots represent spurious attractors, and the blue lines represent the recall trajectories of random vectors.

The results from the Monte Carlo simulation are shown in Figure 4.14 and 4.15. Figure 4.14a presents the average learning time as a function of the number of inputs for both BAM and FEBAM. Results indicated that for both binary and bipolar encoding, learning times increased as the number of inputs increased. However, the rate of increase in learning was nonlinear and tended to decelerate as the number of inputs became larger. Irrespective of the model, binary encoding consistently took about three times longer to learn than bipolar encoding. For instance, in auto-

association task with 25 inputs, the average learning time for bipolar encoding was approximately 115 epochs in the FEBAM and 150 in the BAM, while for binary encoding, it was around 350 epochs for the FEBAM and 450 epochs for the BAM. Figure 4.14b shows the recall performance, showing 100% for all models under each condition, meaning the input patterns were learned for each task and with each model and encoding type.

Figure 4.14c shows the relationship between the number of inputs and the average proportion of spurious attractors (number of spurious vectors recalled / total number random recall vectors) across 100 trials in both BAM and FEBAM models, according to binary and bipolar encoding. Generally, as the number of inputs increased, the proportion of spurious attractors also increased. However, there were notable differences based on the encoding type. The BAM with bipolar encoding exhibited 50% spurious attractors between 1 and 15 inputs, followed by a sudden jump to nearly 100% at 50 inputs. In the case of the BAM with binary encoding, a "U-shaped" curve was observed, with approximately 70% to 80% spurious attractors at 1 and 2 inputs, followed by a sharp decrease to around 15% proportion between 3 and 15 inputs, and a gradual increase to nearly 100% at 50 inputs. For the FEBAM with bipolar encoding, the proportion of spurious attractors closely mirrored the curve seen in the bipolar BAM. However, in the binary FEBAM, there was no "U" shaped curve. Instead, the proportion of spurious attractors remained low, below 1%, until around 15 inputs, after which it sharply increased to nearly 100% as the number of inputs increased towards 50.

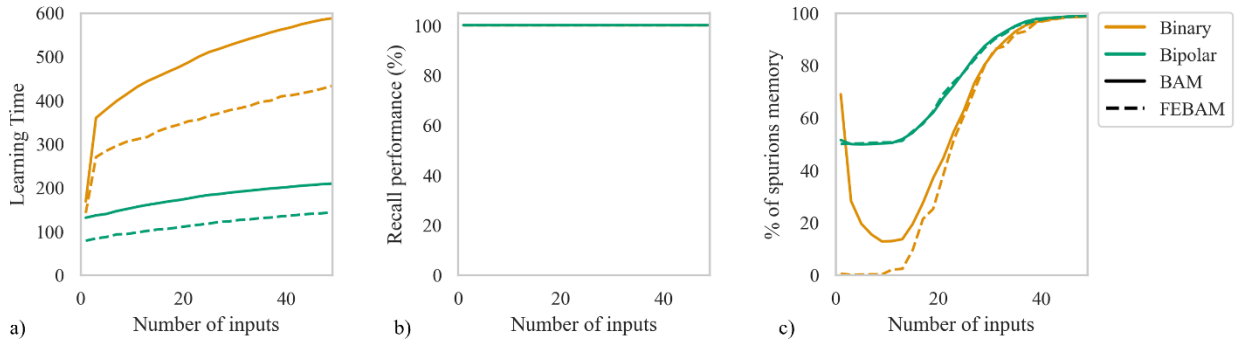

Fig. 4.14. The (a) learning time in number of epochs, (b) recall performance, (c) spurious recall according to the number of inputs in the auto-association task when using bipolar encoding (green) compared to binary (yellow) with the BAM or FEBAM (line style).

The average recall performance under the noisy pixel flip task is shown in Figure 4.15. Results show the traditional graceful degradation of recall performance with increased noise found in BAMs. In general, the higher the number of inputs, the lower recall performance under noise. For both the FEBAM and BAM, binary encoding showed equal or superior recall performances on average for all input number conditions.

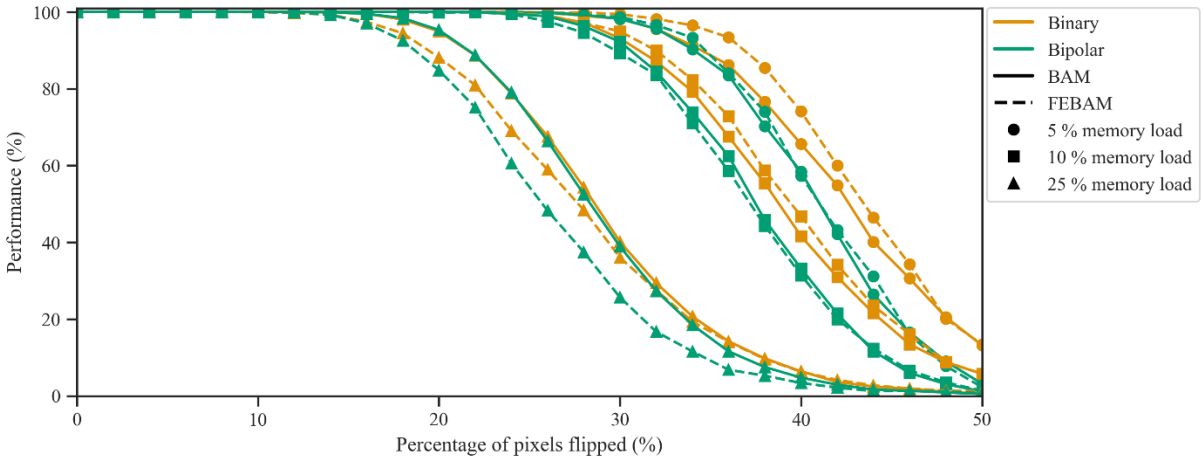

Fig. 4.15. Noise recall performance according to encoding type (colour) and number of inputs (marker style).

4. 4. 3. Simulation I - Discussion

This simulation provided insights into the learning performance using the new binary transmission function. Overall, for both supervised (BAM) and unsupervised (FEBAM) networks, the binary transmission function can learn the same correlated association task as when using the bipolar transmission function but slower (in more epochs). On the flip side, there were fewer spurious attractors and improved recall under noise with binary encoding. This was first visualized with the occurrence of spurious attractors during the learning process of a 3-dimensional task. The modified BAM with binary encoding exhibited a lower count of spurious attractors (1) than binary encoding (3). Notably, the three spurious attractors in bipolar encoding corresponded to the inverse vectors and these were eliminated when using binary encoding, with the sole remaining spurious attractor being the zero vector. Notably, in the BAM, recall of the zero-vector attractor occurred approximately 15% of the time, while in the FEBAM, it occurred less than 1% of the time.

While the 3-dimensional task served its purpose for visual illustration, the Montecarlo simulation revealed more detailed findings as it used higher-dimensional inputs to explore learning time, the prevalence of spurious attractors, and recall performance under noise. Results showed that under both binary and bipolar encoding, the task was fully learned regardless of the number of inputs. The learning time was between 2x and 3x longer with binary encoding than bipolar, which goes in line with previous findings (Kosko, 1988). However, the proportion of spurious attractors was much lower for binary encoding, especially below 40 inputs. One effect of this was that the performance under noisy recall was better under binary encoding. This is due to the network having less spurious attractors meaning a higher chance of a noisy input falling on the correct attractor. Interestingly, when learning only a few numbers of inputs (1-3), the proportion of spurious attractors with the BAM using binary encoding is very high. This results in a “U”

shaped curve. This can be explained by looking at the type of attractors. There are three possible types of spurious attractors and Figure 4.16 shows a visual example of each type of spurious attractor: a distorted or random vector (Figure 4.16b), a complement (inverse) vector (Figure 4.16c), and the zero vector (Figure 4.16d).

Fig. 4.16. Examples of recalled patterns shown as 10 by 10 grid where white pixels represent values of 0, black pixels values of 1. Featuring (a) the original input representing the letter "L", and spurious recall that is b) a complement (inverse) vector, c) a distorted input, and d) a zero vector.

Since in 3-dimensions the only spurious attractor present with binary encoding was the zero vector, results from Figure 14c were modified by removing the presence of the zero attractor amongst the spurious attractor count in Figure 4.17a. The figure reveals that the proportion of spurious attractors remains 0% until 15 inputs with binary encoding, while the curve remains unchanged for bipolar encoding. To confirm this, figure 4.17b revealed the percentage of the zero vector being recalled. In the BAM, the zero vector is recalled 70% of the time when only 1 input is learned, 40% when 3 inputs are learned, around 10% when 15 inputs are learned, and it is not recalled when the number of inputs learned are over 25. Hence, the "U" shaped curve observed in the binary BAM is attributed to the zero vector being the only spurious attractor. Whereas in the bipolar BAM, there are complement and distorted attractors present.

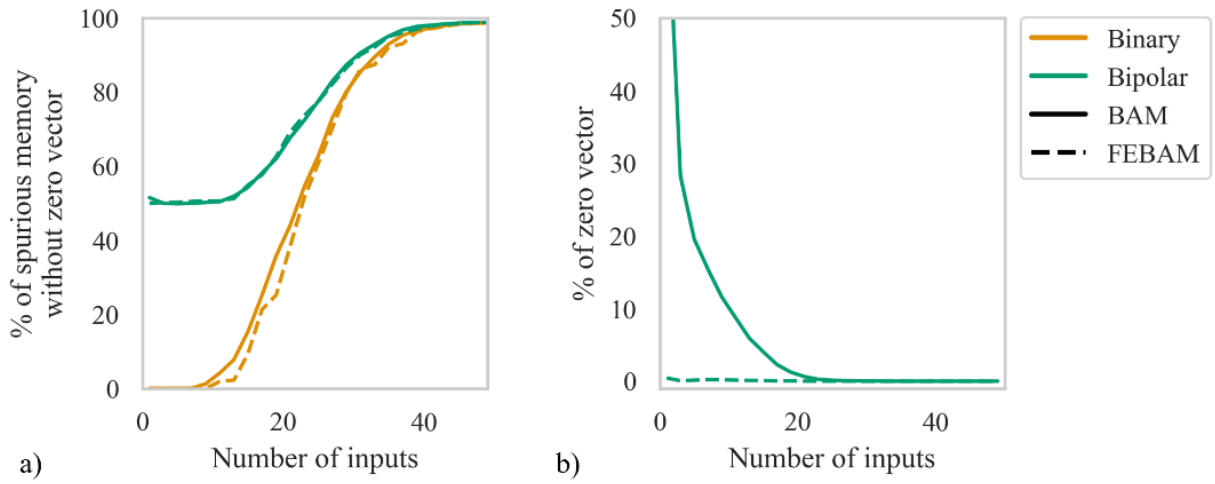

Fig. 4.17. The proportion of spurious attractions for each memory load according to encoding type (colour) and learning parameter, η, (linestyle).

Results from the previous section confirmed that the binary transmission function retains the associative learning capabilities of bipolar encoding and even enhances recall by minimizing spurious attractors, with the primary trade-off being increased learning time. From these findings, two key properties emerge: firstly, zero indeed acts as an attractor in the system, corroborating the theoretical analysis; secondly, the network is capable of yielding a response that consists solely of a zero vector. This property could be leveraged for various cognitive tasks. Consequently, the subsequent section explores additional properties such as one-shot learning and inhibition in the context of enabling multi-network architectures.

4.5 Simulation II – One-shot learning

One of the key findings from the preceding section was that using a binary transmission function enhanced performance but required more epochs for learning. To accelerate learning, one method could be to adjust the learning parameter to an optimal value that minimizes the number of epochs needed. However, this method would be time-consuming, wouldn't ensure faster learning and would focus more on optimization rather than cognitive processes. Moreover, raising the learning parameter might lead to quicker convergence of weights, but often at the expense of performance, as fewer inputs are effectively stored and input attractors may not form correctly, leading to interference. This issue of prolonged learning implies that in binary encoding, the BAM must process the inputs more frequently compared to bipolar encoding, where each epoch involves presenting the network with each input once. A potential solution to enhance learning speed is to implement a mechanism that allows more frequent input presentation in the binary BAM. In ANNs, memory replay has been employed to expedite learning among other applications. Replay mechanisms are a brain-inspired technique that involve reiterating information to a network, often through a secondary network (Hayes et al., 2021; Roscow et al., 2021; Van de Ven et al., 2020). In the context of this chapter, a dual-BAM system could be effective: one BAM with a high learning parameter serving as Short-Term Memory (STM), and another with a standard parameter as Long-Term Memory (LTM). The STM, with limited memory capacity, could frequently relay any stored input to the LTM, which, though slower, would be able to store more inputs. This dual-network and replay mechanism are akin to complimentary learning systems (Blakeman & Mareschal, 2020; Kumaran et al., 2016; O'Reilly & Norman, 2002; O'Reilly et al., 2014; Sun et al., 2023). Ideally, the STM would complete learning in a single epoch, necessitating only one presentation per input. Setting the learning parameter to its maximum value of 1 could achieve the fastest convergence of weights. However, in bipolar BAMs, even this approach requires several

epochs for input learning. Additionally, there is always a residual retention of old inputs, causing interference when learning new inputs. For an RNAM to function effectively as an STM in a replay system, it would need to overwrite or eliminate any previously stored information to prevent interference and ensure clean information replay. This is yet to be tested with binary encoding in the BAM and may be feasible due to the absence of spurious attractors other than the zero vector under a low input load. Therefore, this section will initially conduct a preliminary analysis to investigate one-shot learning in a binary BAM through a simple auto-associative task. Subsequently, it will explore the application of this one-shot learning ability in a multi-network architecture, where a STM composed of multiple one-shot learning BAMs enables a LTM, a standard BAM, to learn from a single presentation of inputs from the environment via a replay system.

4.5.1 Preliminary analysis

4.5.1.1. Procedure. A simple auto-association task was used in this preliminary analysis to examine the one-shot learning ability in the BAM using binary transmission function. The task consisted of learning four 100-dimensional correlated vectors (binary or bipolar) representing the letters A to D. Learning proceeded as described in section 4.3.4. The learning parameter (η) was set to 1 to ensure maximum weight change and the maximum number of epochs was set to 1, meaning each input was presented only once. Learning and recall error for each input was measured every time a new input was presented. Additionally, the weights were initialized to zero. To measure spurious attractor proportion, 1000 random vectors were generated and used in recall, as in section 4.4.

4.5.1.2. Results. The preliminary analysis revealed that with the binary transmission function, the BAM could learn each input individually within a single weight cycle, or one

presentation of the input. However, when a new input was introduced, the previous input was overwritten and forgotten, while the new input was learned immediately, as evidenced by Figure 4.18a. This figure shows that the recall error for each input dropped to zero only during its respective learning cycle. By the end of the epoch, after all inputs were presented, only the last input "D" remained stored in the memory. Figure 4.18b indicates that following the epoch, the only input recalled was "D," with no spurious attractors present. Consequently, the one-shot learning capability of the network with the learning parameter set to 1 is limited to one input per epoch. Nevertheless, the binary transmission function prevented the formation of spurious attractors, ensuring that the memory was effectively reset with each new input. This contrasts with other one-shot learning RNAMs which may hold a greater capacity but tend to retain remnants of previous inputs, thereby distorting the learning of new data. Consequently, the subsequent part of this section explored a system of multiple one-shot learning networks linked together, called the STM. This arrangement aimed to increase overall storage capacity and incorporate a replay mechanism for enhancing memory of a standard BAM (the LTM).

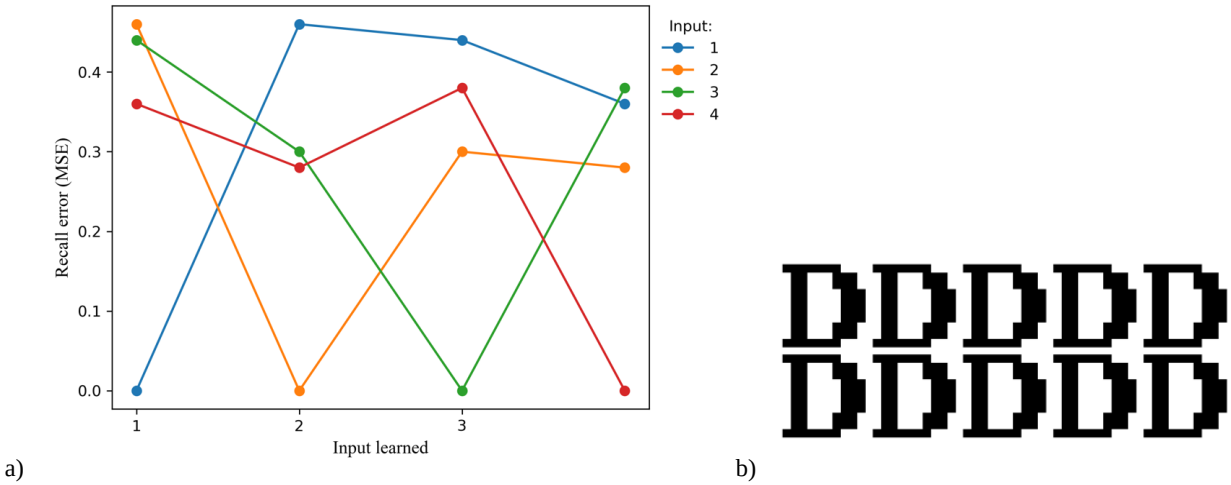

Fig. 4.18. (a) Recall error (MSE) of each input in terms of which input was currently presented and (b) examples of recall in the random vector recall after learning the letter “D”.

4.5.2 Architecture

A multi-network architecture is used and shown in Figure 4.19a. The architecture contains two modules, a Short-Term-Memory (STM) and a Long-Term-Memory (LTM). Overall, the STM will receive and learn a set of inputs one by one from the environment. It will then replay this input set to the LTM for its consolidation. As such, the LTM will have learned the inputs even if the inputs were presented by the environment once. The LTM is simply a BAM as described in Section 4.3 with its learning parameter set to 0.01 as the previous simulation. The architecture shown in Figure 4.19. The STM is a series of *b* BAMs connected in a loop. Each BAM has its learning parameter and max number of epochs set to one. The first BAM, B_1, receives an input from the environment, $\mathbf{x}_{0[0]}$, and learns it in one-shot. Its output, $\mathbf{x}_{0[1]}$, is given to the second BAM, B_2. The output of B_2, $\mathbf{x}_{0[2]}$, is then given to a third BAM, B_3, and so one until the final BAM, B_b, generated the final output, $\mathbf{x}_{0[b]}$, which is given to the LTM to learn and given back to B_1. In this way, the final output is equal to the initial input, $\mathbf{x}_{0[0]} = \mathbf{x}_{0[b]}$. However, this process is continuous, meaning after the first BAM received the first input, the second BAM receives the second input, $\mathbf{x}_{1[0]}$, and this cascades down the next BAMs and eventually B_b will learn it and generate the outpu, $\mathbf{x}_{1[b]}$, which will be given to the LTM and fed back to B_1. This process continues for *b* inputs until B_b generates $\mathbf{x}_{b[b]}$. At this point, the output of the previous BAM, B_{b-1} will be the first input again, $\mathbf{x}_{0[b-1]}$. From here, the STM can infinitely recall the input set, even if only received from the environment once. In other words, once inputs are presented from the environment, the STM can loop these and present them over and over to the LTM. The STM chain will be able to hold *b* inputs. Here, for simplicity, *b* was set to 10 as this was the number of inputs in a task.

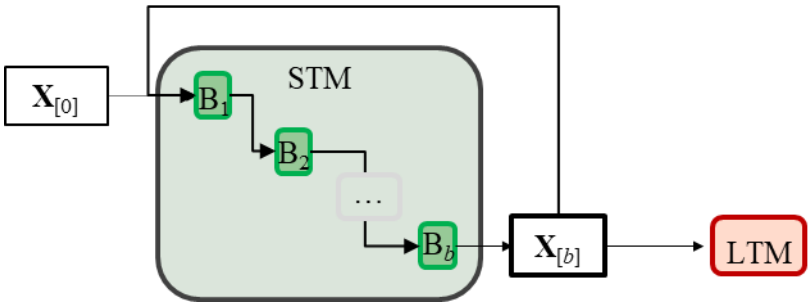

Fig. 4.19. The architecture used in this simulation, where a Short-Term-Memory (STM), in gray, replays the input set **X**, to the Long-Term-Memory (LTM), in orange. The LTM is a Bidirectional Associative Memory (BAM). In the grey box is a detailed breakdown of the STM network, where *b* BAMS, noted as **B**, cycle the inputs and to replay them to the STM.

4.5.3 Procedure

To test the replay system and the architecture, an association task was used consisting of ten 100-dimensional binary vectors, representing the digits 0 to 9, as shown in Figure 4.20.

Fig. 4.20. The task set used to consist of 10 binary input vectors of 100 dimensions shown in 10 by 10 format, where black pixels represent the value of 1 and white pixels the value of zero.

In both the STM and LTM, weights were initialized to zero. In the LTM, the learning parameter was set to 0.01 and the minimum error was set to 10^{-6}.

Learning in the model proceeded in the following way:

1. Initialize weights of STM and LTM to 0.
2. Cycle each input $\mathbf{x}_{[0]}$ ($\mathbf{x}_{[0]} = [\mathbf{x}_{1[0]}, \mathbf{x}_{2[0]}, \ldots \mathbf{x}_{b[0]}]$) one by one in the STM to obtain all STM outputs $\mathbf{x}_{[b]}$ ($\mathbf{x}_{[b]} = [\mathbf{x}_{1[b]}, \mathbf{x}_{2[b]}, \ldots \mathbf{x}_{b[b]}]$) where $b = 10$.
3. Give STM outputs, $\mathbf{x}_{[b]}$, one by one to the LTM to learn.
4. Evaluate recall error in the LTM following section 4.3.5 using $\mathbf{x}_{[0]}$.
5. Cycle $\mathbf{x}_{c[b]}$ in the STM loop to obtain $\mathbf{x}_{c+1[b]}$ once again.

6. Repeat steps 3 to 5 until the minimum error is reached.

To facilitate a comparison, the LTM (a binary BAM) was used to learn the task the standard way, following the methodology outlined in section 4.3.4. The recall error across all inputs served as the primary metric for assessing learning performance. Finally, recall under noised was measured in the LTM using the pixel flip tasked described in section 4.3.

4.5.4 Results

The recall error (MSE) for the STM-LTM model with binary encoding is shown in Figure 4.21a. As show, after around 60 STM recalls, the LTM reached 0 recall error. In comparison, it would take the LTM around 60 epochs to reach a recall error of zero (Figure 4.21b). Recall under noise was measured after learning in both situations in the LTM and results were identical, so they were omitted. As such, the replay mechanism did not negatively affect recall performance under noise.

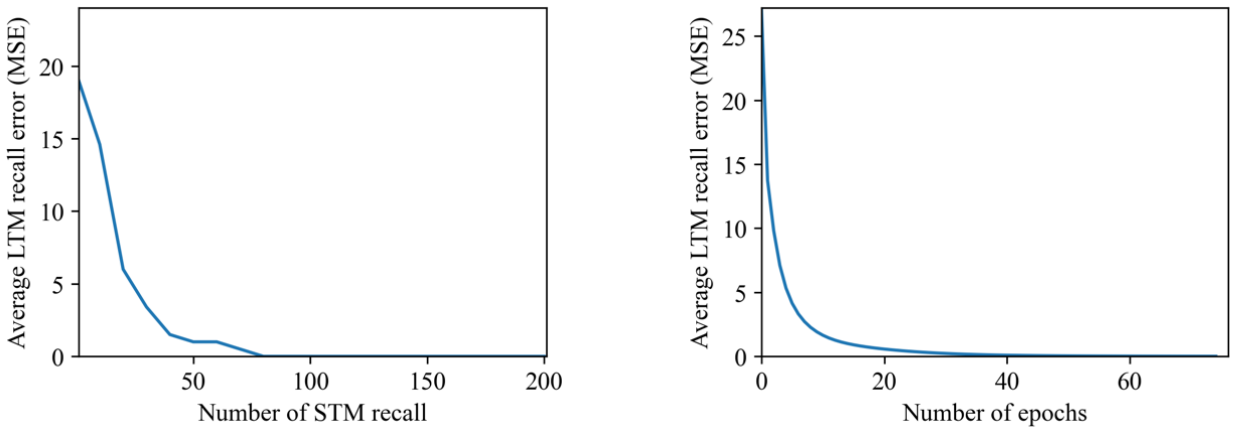

Fig. 4.21. Recall error (MSE) for the (a) STM-LTM in binary encoding, and (b) LTM only.

4.5.5 Discussion

This section showed that the binary transmission function enabled one-shot learning, which was capable of processing a single input without generating any spurious attractor, a marked

improvement over bipolar encoding and other RNAMs, which would retain at least the complement attractor. This one-shot learning was applied as a replay mechanism, allowing a LTM to learn a set of inputs, each presented only once by the environment, through repeated exposure by an STM. Normally, an LTM would require about 75 epochs to learn, corresponding to 75 instances of each input being presented by the environment. In contrast, the STM-LTM architecture allowed learning when inputs were presented by environment only once yet achieved similar learning performance through approximately 75 recalls by the STM. The replay system essentially reduced the number of epochs needed by the LTM to learn to effectively one.

Moreover, the structure of the STM network played a critical role; the number of subnetworks, *b*, within the STM dictated the length of the learning chain. This simulation limited the chain to the size of the task to serve as a proof of concept. For tasks with more inputs, the chain could still remain at 10 subnetworks ($b = 10$), as an 11^{th} input would simply overwrite the older input in the subnetwork chain. Alternatively, the task could be segmented into separated sequences equal to the number of subnetworks. Future research could examine different number of subnetworks, *b*, and examine this replay system in more complex tasks and how replay systems affect human memory (Kaefer et al., 2022; Ólafsdóttir et al., 2018). In fact, the current architecture could be examined as a sort of cognitive load capacity of working memory (Miller, 1956; Sweller, 2011; Sweller et al., 2019). Additionally, future research could look into expanding the capacity of the system to handle more than one input simultaneously. In the next sections, other new properties obtained with binary encoding will be assessed, particularly taking advantage of the absorbent nature of zero attractor as an inhibitory mechanism.

4. 6. Simulation III – Gating

Given that zero represents a stable fixed point in the transmission function, it is feasible for a BAM to learn to associate inputs with the zero vector in an association task. This ability could enable the network to use the zero vector as a conditional inhibitor by applying the Hadamard product to any incoming input. Such a mechanism could serve multiple purposes. For instance, it could be used to regulate information received by another network, by selectively reducing to zero any unwanted input, which would not impact learning. It could also be used to selectively direct the flow of information between different networks, acting as a gating mechanism. To test both these possibilities, this simulation implemented two multi-network architectures comprising two types of networks: a Modulator and a Learner. The first architecture used one Modulator and Learner to examine if the Modulator could selectively inhibit specific inputs from being learned by the Learner. The second architecture used two Learners to examine if the Modulator could direct inputs to either Learners.

4.6.1 Architecture

Both the Modulator and Learner are BAMs using the new binary transmission function as described in section 4.3. Two different architectures were tested either having one Learner or two Learners. In each case, two sets of association tasks were used, an instruction set for the Modulator to find decision vectors according to context vectors, and a problem set used to test the gating mechanism.

4.6.1.1. Architecture 1. The first architecture is shown in Figure 4.22a. two sets of associations were used, an instruction set for the Modulator and a task set for both networks. The Modulator was trained to associate specific input types with either a zero vector or a one vector

following the instruction set. Upon encountering new information in an association task, the Learner network would be dictated by the recalled decision vector from the Modulator—either the zero vector or the one vector. This vector would be multiplied elementwise (Hadamard product) to the incoming stimuli from the Problem set before its transmission to the Learner network. A recalled zero vector from the Modulator would result in the incoming input being nullified (multiplied by zero), effectively inhibiting any learning activity in the Learner. Conversely, if the one vector was recalled, the incoming input would remain unchanged due to multiplication by a vector of ones, permitting the Learner network to process and learn from the incoming input.

4.6.1.2. Architecture 2. A second architecture was created to further investigate the gating mechanism with a Modulator and two Learners ($Learner_1$ and $Learner_2$), as shown in Figure 4.22b. Once again, each network is a BAM as described in section 4.3. The Modulator functions in a similar way as described in section 4.6.1.1, learning an instruction set using context and decision vectors as inputs. However, in this case the decision vectors are not the one or zero vector. Effectively, the decision vectors, $\mathbf{q}$, are broken down into sub decision vectors, $\mathbf{q}_1$ and $\mathbf{q}_2$, which will be used on the inputs from the problem set, $\mathbf{x}$, before the respective Learners, i.e. a Hadamard product $\mathbf{x} \odot \mathbf{q}_1$ for $Learner_1$ and $\mathbf{x} \odot \mathbf{q}_2$ for $Learner_2$.

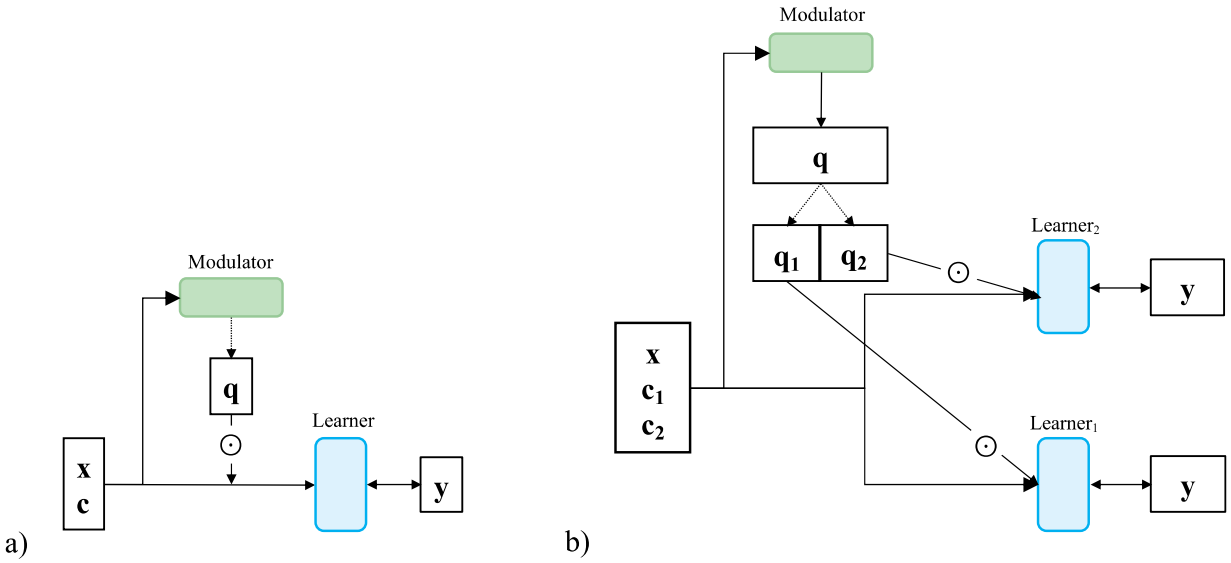

Fig. 4.22. Multi-network architectures used in this simulation where Modulator and Learners are individual Bidirectional Associative Memories (BAM). In (a), there is a modulator and a Learner. The input pattern, **x**, concatenated to a context, **c**, is presented to the Modulator which recalls the decision vector, **q**. The Hadamard product is then perform between the decision vector and the input (**x**⊙**q**) before being presented to the Learner, which will learn to associate it to the corresponding input **y**. The second architecture is proposed in (b) where a Modulator controls the information received by two Learners. The Modulator which recalls a decision vector **q** that will be divided in half into two more decision vectors ($\mathbf{q}_1$ and $\mathbf{q}_2$) used on the inputs **x** before being presented to the Learners ($\mathbf{x}\odot\mathbf{q}_1$ for Learner_1 and $\mathbf{x}\odot\mathbf{q}_2$ for Learner_2)

4.6.2 Inputs

Two association sets were used involving binary vectors, the problem set, and the instruction set. The problem set was an auto-association task which had to be learned by the Learner(s) network(s). Each input in the problem set was concatenated between one of two contexts, “A” or “B”. The same problem set was used for both architectures. The instruction set was a hetero-association task used to train the Modulator to selectively recall masks according to contextual information in the input, which were used to inhibit or allow input in the problem set prior to reaching the Learner(s). In the instruction set, there were two context vectors, **c**, and two decision vectors, **q**. The context vectors were crucial to allow the Modulator to identify which decision vectors to recall and then

use on the inputs of the problem set via Hadamard product. Different instruction sets were used for both architectures.

4.6.2.1. Problem set. This was auto-associative task using randomly generated 100-dimensional binary input vectors. Randomly generated inputs were used to test the process multiple times. The random vectors were equally divided into two groups by being concatenated with either two 100-dimensional context vectors: one representing the number “1” and the other the number “2” in a 10 by 10 format. This approach aligns with prior research that incorporates multi-dimensional patterns as contextual cues (Rolon-Mérette et al., 2018). An example of the task set is shown in Figure 4.23.

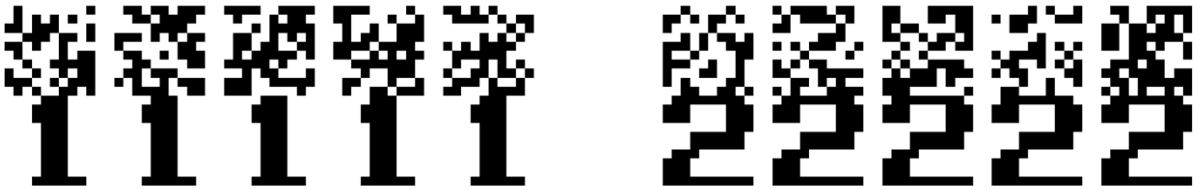

Fig. 4.23. Example of **x** inputs 200-dimensional vectors belonging to the problem set, where white pixels represent a value of 0 and black pixels a value of 1, visualised in a 20 by 10 format. The upper half (first 100 values) are the randomly generated vectors, and the lower half (second 100 values) are the context vectors “1” and “2”.

4.6.2.2. Instruction set for architecture 1. In the first architecture, two instruction sets were created. In instruction set A, the context vector with a "1" was to be associated to a 200-dimensional vector of ones and the context vector with a "2" was associated with a 200-dimensional vector of zeros. The decision vectors were of dimension (length) 200 to match the inputs with context in the problem set. Similarly, each context was concatenated with a 100-dimensional zero vector serving as a “filler” pattern to maintain dimensional consistency between instruction and task set. In instruction set B, the associations were inversed, as context “1” were associated to the zero vector and context “2” were not associated to the one vector. Each of these

instruction sets are shown in Figure 4.24. This was done to evaluate if the Modulator could learn different instruction sets to reverse its impact on the Learner.

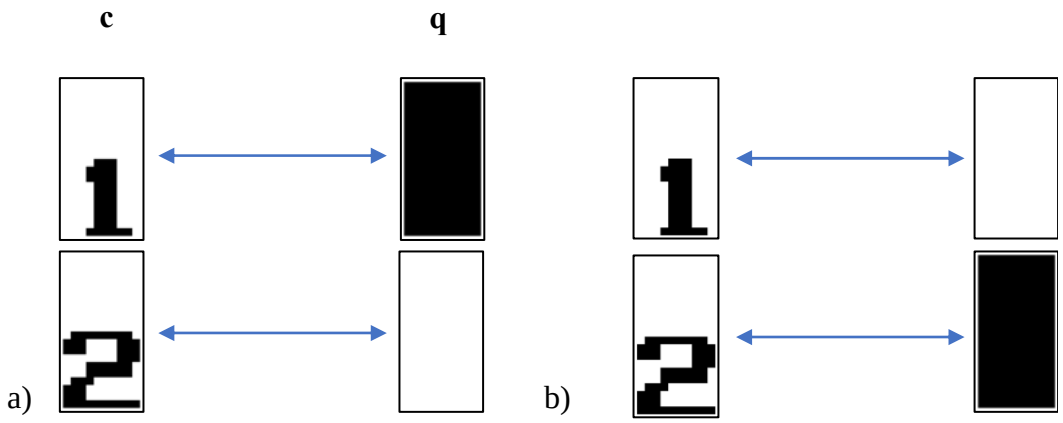

Fig. 4.24. The two instruction sets used by the Modulator. Each instruction set has two associations, where white pixels represent a value of 0 and black pixels a value of 1. The two **c** inputs (patterns on the left of the blue bidirectional arrows) are the context vectors of dimension 200, visualised in a 20 by 10 format. The two **q** inputs (patterns on the right of the blue bidirectional arrows) are 200-dimensional decision vectors visualised in a 20 by 10 format.

4.6.2.3. Instruction set for architecture 2. For the second architecture, the instruction set had the same context vectors as input, but the decision vectors were two 400-dimensional decision vectors, with half of their values being 0 and half of their values being 1, as shown in Figure 4.25. This allowed the decision vectors, **q**, to be divided equally in two sub-vectors, $\mathbf{q}_1$ and $\mathbf{q}_2$, one for each Learner. The goal was to allow Learner_1 to learn inputs with context 1 but not context 2 and vice-versa for Learner_2.

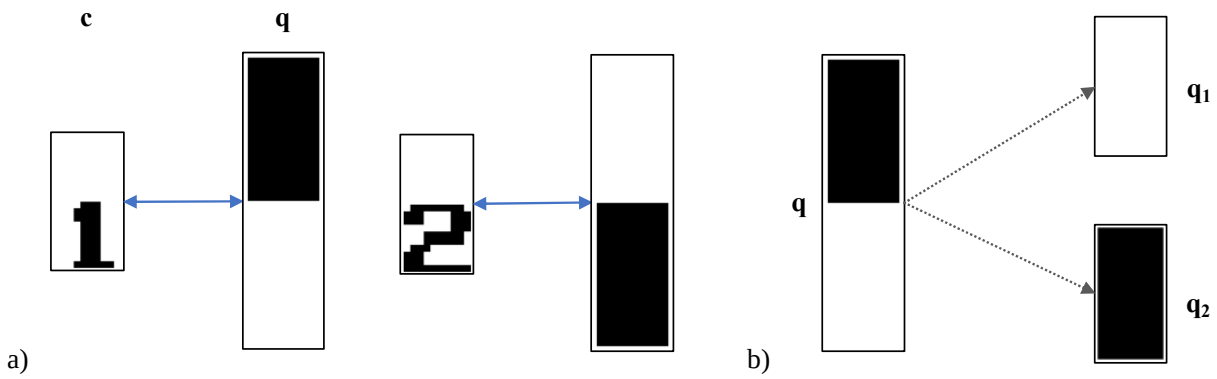

Fig. 4.25. The instruction set for the second architecture with two learners (a) the two associations of the instruction set used in the Modulator, where white pixels represent a value of 0 and black pixels a value of 1. The two **c** inputs (patterns on the left of the blue bidirectional arrows) are the context vectors of dimension 200, visualised in a 20 by 10 format. The two **q** inputs (patterns on the right of the blue bidirectional arrows) are 400-dimensional decision vectors visualised in a 40 by 10 format. (b) During the task set, once the decision vectors **q** are recalled, they are divided into sub decision vectors $\mathbf{q}_1$ and $\mathbf{q}_2$.

4.6.3. Procedure

As mentioned, both the Learner and Modulator networks were BAMs using the binary transmission function. To account for dimension size, the weights were initialized to zero and the learning parameter was set to 0.01. The number of recall cycles in each network was set to a maximum of 100 and the maximum number of epochs was set to 50 or a minimum error of 10^{-6}. The simulation for the first architecture was conducted in following way:

1. Initialize weights of Modulator and Learner to 0.
2. Create the instruction set A with inputs $\mathbf{c}_A$ and $\mathbf{q}_B$ as described in section 4.6.2.2.
3. Learning of the instruction set in the Modulator following section 4.3.4.
4. Create the problem set with randomly generated inputs following section 4.6.2.1.
5. Randomly select an input from the problem set to obtain $\mathbf{x}_{[0]}$. Set $\mathbf{y}_{[0]} = \mathbf{x}_{[0]}$.

6. Present $\mathbf{x}_{[0]}$ to the modulator and perform recall following section 4.3.5 to obtain the decision vector, $\mathbf{q}_{[0]}$.
7. Perform an element-wise multiplication (Hadamard product) of the decision vector, $\mathbf{q}_{[0]}$, with the input $\mathbf{x}_{[0]}$ ($\mathbf{x} = \mathbf{x} \odot \mathbf{q}$) and then present the resulting vector, $\mathbf{x}$, to the Learner.
8. Cycle through the Learner using the binary transmission function (Equations 4.14a and 4.14b) to obtain the outputs $\mathbf{x}_{[1]}$ and $\mathbf{y}_{[1]}$.
9. Update the weights of the Learner according to the learning rule (Equations 4.15a and 4.15b) using $\mathbf{x}_{[0]}$, $\mathbf{x}_{[1]}$, $\mathbf{y}_{[0]}$ and $\mathbf{y}_{[1]}$.
10. Repeat steps 5 to 9 until the maximum number of epochs is reached.
11. Change the instruction set to B using inputs $\mathbf{c}_{\mathbf{A}}$ and $\mathbf{q}_{\mathbf{B}}$ and repeat steps 2 to 10.

It is of note that the weights of the Modulator and Learner networks were only initialized once and never manipulated again, meaning that the weights from the end of one instruction set are used when another instruction set is introduced. The simulation procedure for the second architecture was as follows:

1. Initialize weights of Modulator and Learner to 0.
2. Create the instruction set with inputs **c** and **q** as described in section 4.6.2.3.
3. Learning of the instruction set, inputs **c** and **q**, in the Modulator following section 4.3.4.
4. Create the problem set with randomly generated inputs following sections 4.6.2.2.
5. Randomly select an input from the problem set to obtain $\mathbf{x}_{[0]}$. Set $\mathbf{y}_{[0]} = \mathbf{x}_{[0]}$.

6. Present $\mathbf{x}_{[0]}$ to the modulator and perform recall following section 4.3.5 to obtain the decision vector, $\mathbf{q}_{[0]}$.
7. Separate the first half of pixels in the decision vector, $\mathbf{q}_{[0]}$, into sub decision vectors, $\mathbf{q_1}_{[0]}$ and $\mathbf{q_2}_{[0]}$.
8. Perform an element-wise multiplication (Hadamard product) of the decision vector, $\mathbf{q_1}_{[0]}$, with the input $\mathbf{x}_{[0]}$ to obtain $\mathbf{x_1}_{[0]}$ ($\mathbf{x_1} = \mathbf{x} \odot \mathbf{q}_2$) and then present the resulting vector to the Learner$_1$.
9. Perform an element-wise multiplication (Hadamard product) of the decision vector, $\mathbf{q_2}_{[0]}$, with the input $\mathbf{x}_{[0]}$ to obtain $\mathbf{x_2}_{[0]}$ ($\mathbf{x_2} = \mathbf{x} \odot \mathbf{q}_2$) and then present the resulting vector to the Learner$_2$.
10. Cycle through the Learner$_1$ using the binary transmission function (Equations 4.14a and 4.14b) to obtain the outputs $\mathbf{x_1}_{[c]}$ and $\mathbf{y_1}_{[c]}$.
11. Cycle through the Learner$_2$ using the binary transmission function (Equations 4.14a and 4.14b) to obtain the outputs $\mathbf{x_2}_{[c]}$ and $\mathbf{y_2}_{[c]}$.
12. Update the weights of the Learner$_1$ according to the learning rule (Equations 4.15a and 4.15b) using $\mathbf{x}_{[0]}$, $\mathbf{x_1}_{[0]}$, $\mathbf{y}_{[0]}$ and $\mathbf{y_1}_{[c]}$.
13. Update the weights of the Learner$_2$ according to the learning rule (Equations 4.15a and 4.15b) using $\mathbf{x}_{[0]}$, $\mathbf{x_2}_{[0]}$, $\mathbf{y}_{[0]}$ and $\mathbf{y_2}_{[c]}$.
14. Repeat steps 5 to 14 until the maximum number of epochs is reached.

4.6.4 Results

4.6.4.1 Results for architecture 1. For the first architecture, Figure 4.26 shows the learning error (MSE) according to epochs for the Modulator for each instruction set. As shown, each instruction set was fully learned, reaching an error of 0 within 20 epochs. This confirmed that the zero vector

could be learned when using the binary transmission function regardless of the input. Recall performance of the decision vectors was 100%. However, it is of note that inputs associated to the zero vector could not be recalled, meaning learning the zero vector results in the loss of bidirectionality of such inputs. This was expected as **V** weights in the Modulator are not updated by the zero vector.

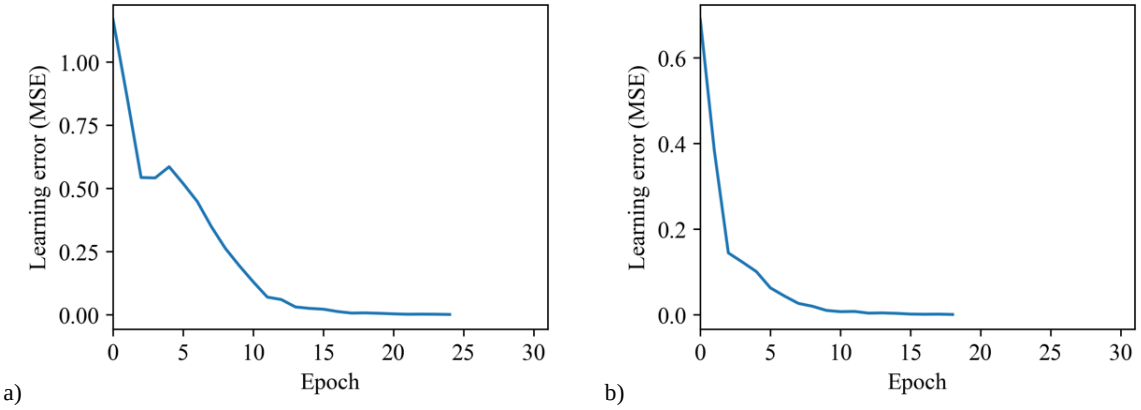

Fig. 4.26. Learning error according to epochs of the Modulator for (a) instruction set A and (b) B.

Figure 4.27 shows the learning error according to epochs of the Learner network for the problem set. There are two lines for learning error, one for inputs with context "1" (blue) and one for inputs with context "2" (orange). The Modulator first learned the instruction set A and at 50 epochs, learned each instruction set B. During the first 50 epochs, Instruction set A was used in the Modulator, and inputs with context "1" were learned as learning error gradually approached zero within 10 epochs. However, inputs with context "2" was not learned, as learning error remained around 0.5 MSE. Recall performance for inputs with context 1 was 100% and for inputs with context 2 it was 0%. During epochs 50 to 100, instruction set B was used, and inputs with context "2" were learned as error gradually approached zero within 20 epochs. However, inputs with context "1" were unlearned, as learning error increased to and remained around 0.5 MSE. Recall performance for inputs with context 1 was 0% while 100% for inputs with context 2.

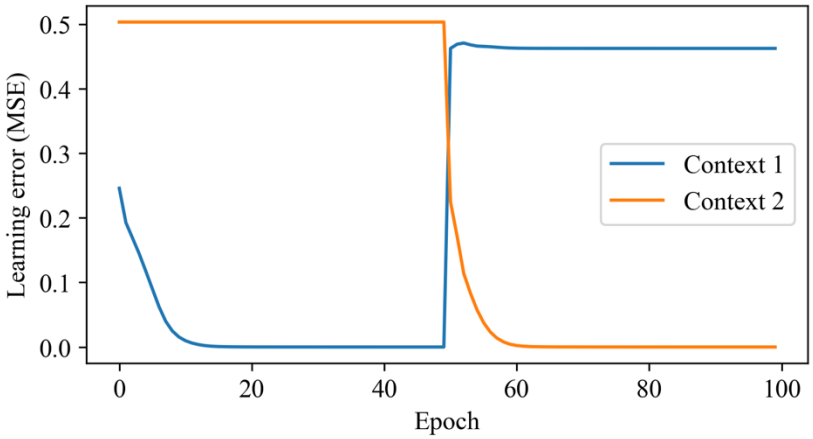

Fig. 4.27. Learning error (in Mean Square error, MSE) according to the number of epochs in the Learner network during the task set. Error is separated according to inputs using context "1" (blue) and "2" (orange).

4.6.4.2 Results for architecture 2. For the second architecture, the Modulator successfully learned the instruction set within 20 epochs, so the learning curve was omitted as it follows the same style of curve as shown in Figure 4.26. Learning error (MSE) as a function of the number of epochs in the networks $Learner_1$ and $Learner_2$ is shown as in Figure 28a and 28b respectfully. The error of inputs with context 1 reached 0 MSE for $Learner_1$ within 20 epochs while it remained around 0.5 for $Learner_2$. The opposite was found for inputs with context 2, as the error reached 0 MSE for $Learner_2$ in around 25 epochs but remained around 0.5 MSE for $Learner_1$.

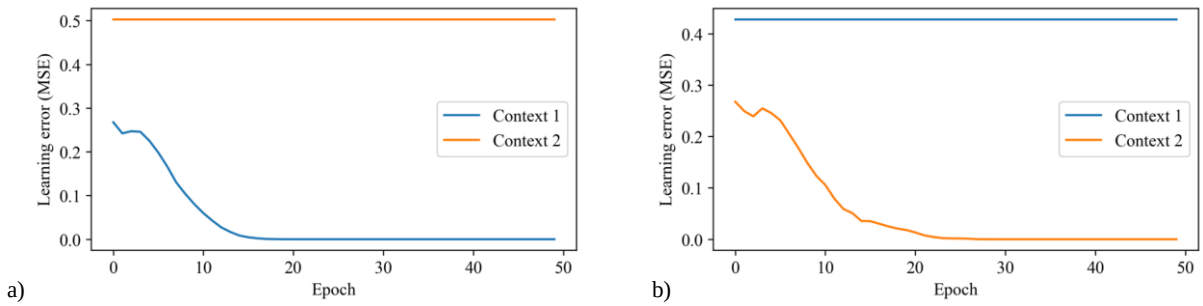

Fig. 4.28. The learning error (Mean square error, MSE) according to the number of epochs for (a) Learner1 and (b) Learner 2. The error is divided by type of contexts for inputs in the task set, where learning error of inputs with contexts "1" is represented by the blue line and error for inputs with context "2" by the orange line.

4.6.3 Simulation II - Discussion

This section explored how a network could learn and use the zero vector as an inhibitory mechanism in architectures involving multiple networks. Initially, it was confirmed that the network, using a binary transmission function, could associate multi-dimensional inputs with the zero vector without affecting its ability to learn associations between nonzero vectors. This result was anticipated because zero is a stable attractor in the binary transmission function but verifying it in a neurodynamic context was nonetheless important. A significant finding was the use of the zero vector as an inhibitor in two different network setups. In the first architecture, involving a Modulator and a Learner, the Modulator was able to selectively recall the zero-vector based on input context and use it to prevent the Learner from processing these inputs. For instance, with a specific instruction set, the Modulator associated 'context 1' with a vector of ones and 'context 2' with a zero vector. During tasks, if 'context 2' was detected in the inputs, the Modulator recalled the zero vector, which, when multiplied by the input (Hadamard product), transformed the input into a zero vector. This effectively blocked the Learner from recognizing the input. Conversely, inputs associated with 'context 1' were unchanged by this multiplication and were learned by the Learner. A switch in the instruction set allowed the Learner to process inputs linked with 'context 2' while forgetting those associated with 'context 1', showing that the zero vector could also facilitate forgetting. In the second architecture, the zero vector was used similarly to act as a gate, controlling information flow between two Learners depending on the context. At the same time, this second simulation showed that decision vectors do not need to be entire vectors of ones or zeros, and that decision vectors could be split up into subcomponents which can be used to inhibit (or not) information to different networks. In both architectures, only two contexts were used, thus the Modulator only had two possible decisions. However, it would be possible to extend the

number of contexts beyond two. Additionally, many-to-one associations could be done. For instance, only but one context would be learned, such as 1 -> 0; 2 -> 1; and 3 -> 0, or alternatively, learn all contexts except for one, such as 1 -> 0; 2 -> 1; and 3 -> 1. Here, the number of contexts, and thus decisions by the Modulator, is restricted to the limitations of the standard BAM, which are the memory load, and that the instruction must be a linearly separable task. However, this could be solved by using the multilayered FEBAM architecture discussed in Chapter 2 and 3 to allow more contexts to be learned and used by the Modulator.

In this simulation, the decision vectors were used as an all-or-nothing mechanism, being either a vector of ones or zeros. The absorbent property of the zero could be used more subtly. To further explore the inhibition property, the next section will utilize decision vectors as partial masking mechanisms, testing their efficacy in a more nuanced control over what the network learns or ignores.

4.7 Simulation IV - Masking

In the previous simulation, the binary transmission function was used to allow the Modulator to conditionally recall the zero vector and use it as an inhibition mechanism to direct the flow to other networks. This can be seen as the zero vector being used as a global mask, to remove the information of the entire pattern. However, partial masks could also be used to inhibit of specific information within an input vector. As such, in this simulation, a variation of the Modulator and Learner architecture is used. The Modulator learns an instruction set consisting in a series of partial binary masks, each representing different features of the input space. The Modulator recalls these masks and applies them to incoming input before they reach the Learner. In this case, the Learner is a FEBAM, an unsupervised version of the BAM which, as shown in Chapter 1 and 2, generates representations with correlations corresponding to the categories in the input (Giguère et al., 2007;

T. Rolon-Mérette et al., 2019; Rolon-Merette et al., 2018). Thus, using the FEBAM will be useful to examine the impact of the partial masks on the generated representations.

4.7.1 Architecture

The architecture is shown in Figure 4.29. The Modulator is a BAM, and the Learner is a FEBAM. Before the task set, the Modulator will learn an instruction set, **q**, which consists of a series of masks, as described in section 4.6.2. During the task set, **x**, the Modulator will recall each mask and apply them to the input by Hadamard product (**x'** = **x**⊙**q**) before the input is given to the Learner. The instruction set learned by the Modulator is connected end to end, where a given mask allows the recall of a subsequent mask, and so on, until the final mask allows the recall of the first mask. This ensures a continuous loop where binary masks are continuously recalled and applied by the Modulator without the need of external presentation or intervention. To examine the effects of the binary masks, the Learner is this time a FEBAM. In this way, the generated representations of the FEBAM, **h**, can be examined for each mask-input combination.

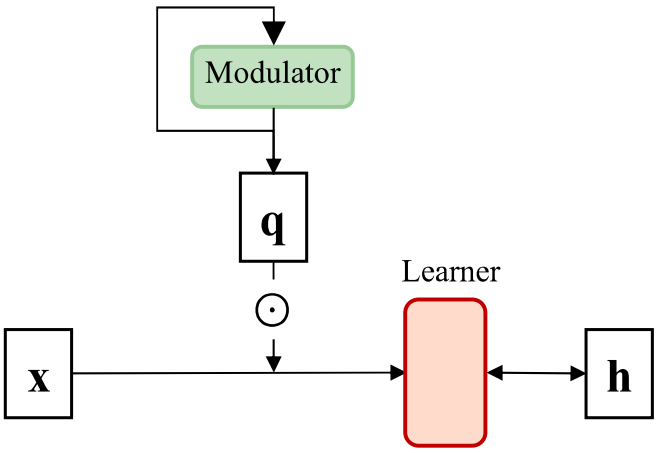

Fig. 4.29. Architecture used in this simulation with a Modulator being a BAM (green) and the Learner a FEBAM (orange). The Modulator first learns an instruction set, **q**, which hetero-association task forming a cycle of associations of binary masks. Then, during the task set, **x**, the Modulator will recall the series of masks which will be applied to the input by Hadamard product (**x**⊙**q**).

4.7.2 Inputs

As with Section 4.6, there are two sets of associations, the instruction set, which is learned by the Modulator, and the task set. The task set is an auto-association task with 26 100-dimensional vectors representing the letters of the uppercase letters of the alphabet when shown in 10 by 10 format, as shown in Figure 4.30.

Fig. 4.30. The task set used to consist of 26 binary input vectors of 100 dimensions shown in 10 by 10 format, where black pixels represent the value of 1 and white pixels the value of zero.

The instruction set is a hetero-associative task consisting of two sequences of seven binary masks. In fact, the second sequence mirrors the first but shifted by one position, forming a cyclical pattern of associations: each mask is associated with its subsequent neighbour, culminating in the final mask linking back to the first. This cyclical relationship is illustrated in Figure 4.31. The initial mask, a vector of ones, enables the entire input to be unchanged and learned by the Learner. The following masks represent horizontal and vertical bars that appear in varied locations within a 10x10 grid. Learning ensures a continuous cycle during recall. During the recall phase, the vector of ones initiates the sequence by triggering the recollection of the subsequent mask—a horizontal bar at the grid's upper edge—which in turn facilitates the retrieval of the succeeding mask, and so forth. The concluding mask, a vertical bar on the right extremity of the grid, is linked back to the vector of ones, thereby sustaining the iterative associative process.

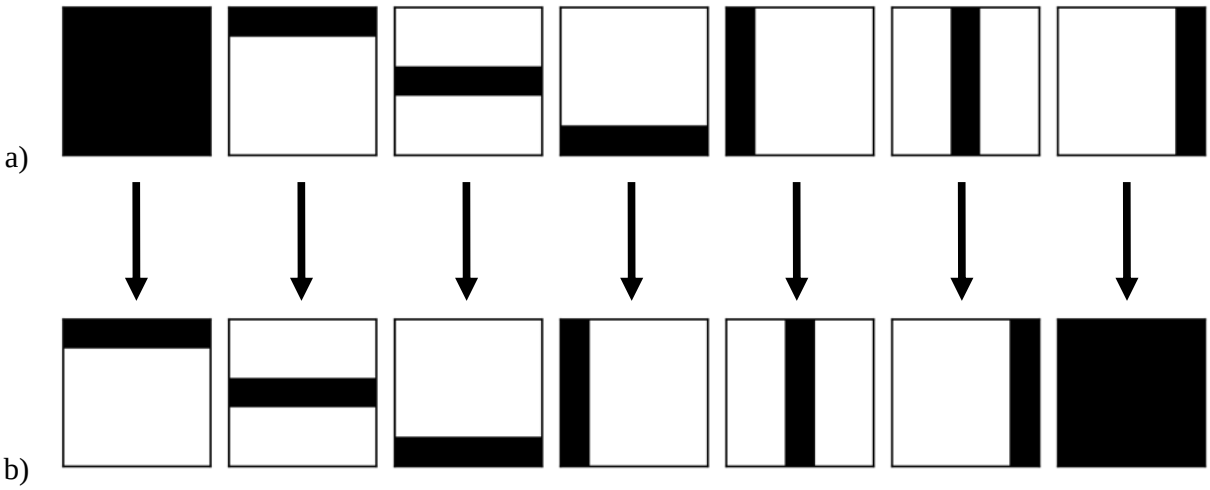

Fig. 4.31. The instruction set composed of two identical series of binary masks, distinguished only by a one-step backward shift in sequence. Inputs from the initial series (a) are paired with the following input in the sequence (b), and the ultimate input—a right-hand vertical bar—is paired with the first mask, the vector of ones.

4.7.3 Procedure

Both the Learner and Modulator network used the binary transmission function and weights were initialized randomly to values between -0.1 and 0.1, as required by the FEBAM. The learning parameter was set to 0.01. The maximum number of epochs was set to 100 or a minimum error of 10^{-6}. The simulation was conducted in following way:

1. Create the instruction set, **q**, as described in section 4.7.2.
2. Learning of the two series in the instruction set, **q**, in the Modulator following section 4.3.4.
3. Create the task set with inputs described section 4.7.2.
4. Perform recall in the modulator using a 100-dimensional vector of ones, $\mathbf{q}_{[0]}$, following section 4.3.5 to obtain the first mask vector, $\mathbf{q}_{[1]}$.
5. Randomly select an input from the task set to obtain $\mathbf{x}_{[0]}$. Set $\mathbf{y}_{[0]} = \mathbf{x}_{[0]}$.

6. Perform an element-wise multiplication (Hadamard product) of the decision vector, $\mathbf{q}_{[1]}$, with the input $\mathbf{x}_{[0]}$ to obtain $\mathbf{x}'_{[0]}$ ($\mathbf{x}' = \mathbf{x} \odot \mathbf{q}$) and then present the resulting vector to the Learner.
7. Cycle through the Learner using the binary transmission function (Equations 4.14a and 4.14b) to obtain the outputs $\mathbf{x}_{[c]}$ and the generated representations, $\mathbf{h}_{[0]}$ and $\mathbf{h}_{[c]}$, following section 4.3.4.
8. Update the weights of the Learner according to the learning rule (Equations 4.15a and 4.15b) using $\mathbf{x}_{[0]}$, $\mathbf{x}_{[0]}$, $\mathbf{h}_{[0]}$ and $\mathbf{h}_{[c]}$.
9. Feedback the mask vector, $\mathbf{q}_{[t]}$, to the Modulator and recall the next mask vector $\mathbf{q}_{[t+1]}$ and repeat steps 6 to 9 for seven times (length of instruction set).
10. Repeat steps 6 to 10 for every input in $\mathbf{x}$.
11. Repeat steps 6 to 11 until the maximum number of epochs is reached.

Here, $\mathbf{y}$ from Section 4.3.4 and equations 4.15a and 4.15b were noted as $\mathbf{h}$ to differentiate the generated representations by the Learner. As with the previous section, it is of note that the weights of the Modulator and Learner networks are only initialized once are never manipulated upon again. To examine representations generated by the Modulator, Pearson correlation was calculated as a measure of similarity. This was calculated with representations generated from inputs within the same mask and between inputs of different masks.

4.7.4 Results

Learning error (MSE) as a function of the number of epochs for the Modulator and Learner is shown in Figure 4.32. Both networks learned successfully. The Modulator successfully learned the instruction set in around 40 epochs as shown in Figure 4.32. The Learner reached the minimum error in around 100 epochs, as shown in Figure 4.32b.

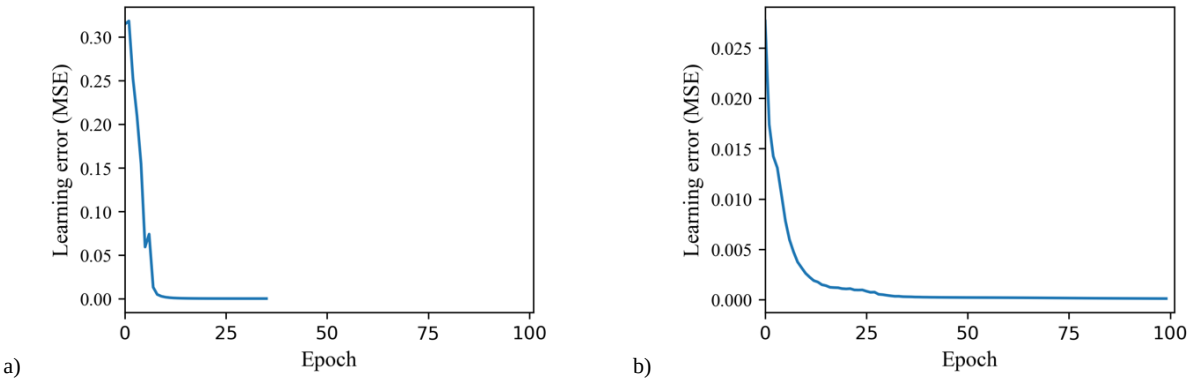

Fig. 4.32. The learning error (Mean square error, MSE) of the (a) Modulator and (b) Learner.

Figure 4.34 shows all generated representations from the 26 inputs according to different masks. As expected, the generated representations followed the structure of the inputs once the masks were applied, i.e. there are the same number of identical representations as the number of identical inputs. Once a mask was applied, if the resulting vector was unique, then a unique representation was generated by the Learner. However, when the resulting masked vector was the same for different raw inputs or masks, the results representations were also the same. Within the same mask, the generated representations were generally similar while they differed significantly from one mask to the next. The representations generated using the vector of ones are all unique, as these correspond to the unchanged letters.

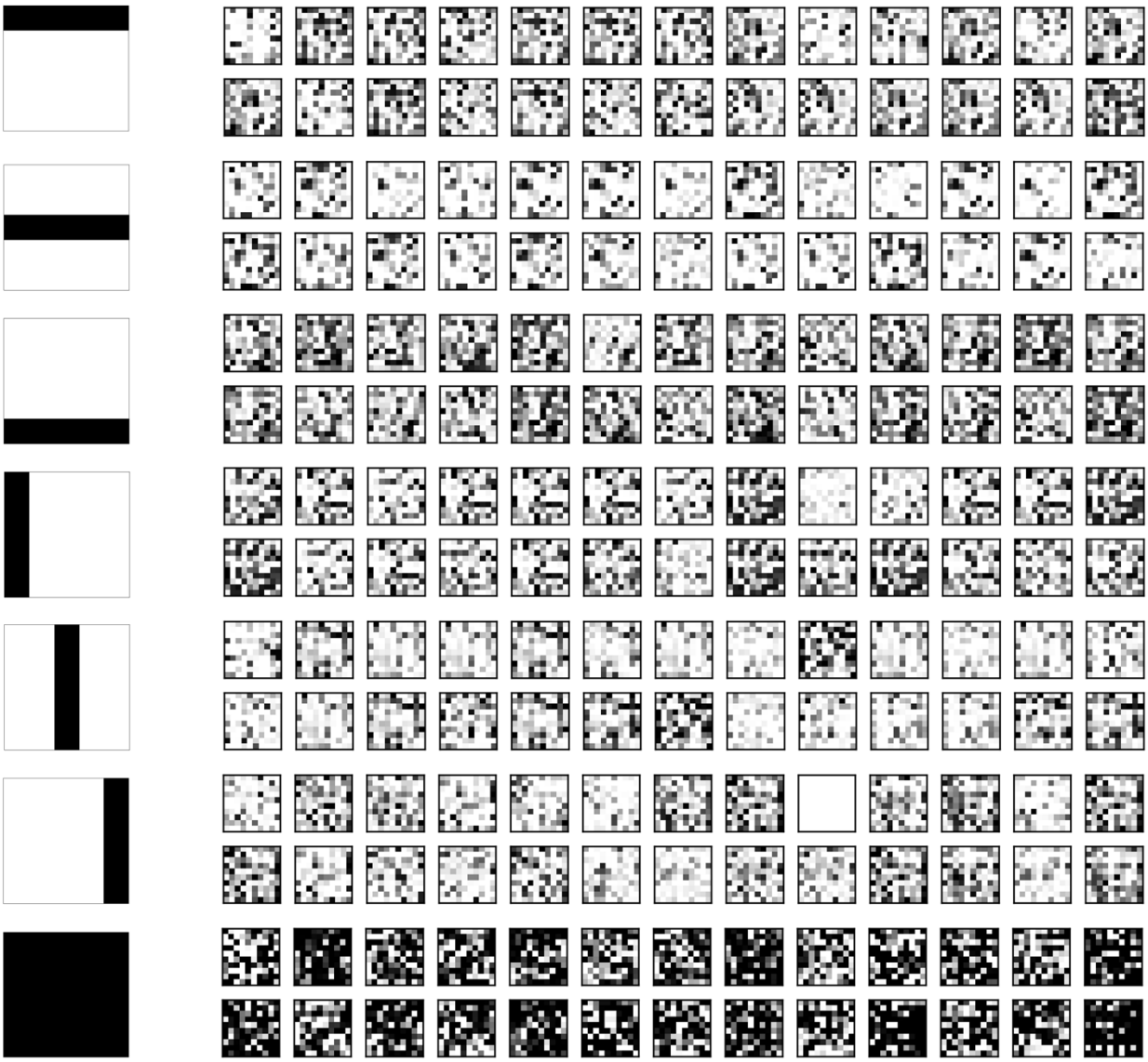

Fig. 4.33. The generated representations by the Learner according to different masks applied by the Modulator. The top row are the representations for letters from A to M, and bottom row letters from N to Z. The bottom row corresponds to the mask of 1 meaning inputs are unchanged and representations were generated based on the raw inputs.

This can also be seen by examining the correlation. The average correlation of generated representations by the Learner according to modulator mask is shown in a correlation matrix in Figure 4.34. This is the average correlation between the representations belonging to each input in the task separated by the type of mask used. For reference, the average correlation between the raw inputs (letters) was 0.16. The correlation of representations from the same mask (the diagonal

in Figure 4.30) was on average 0.1 and higher than the correlation between representations from different masks (rest of matrix) which was on average 0.05. By analyzing correlations of individual representations, some generated representations are quite different (correlation > 0.01) and some are very similar (correlation > 0.9) or exactly the same (correlation = 1).

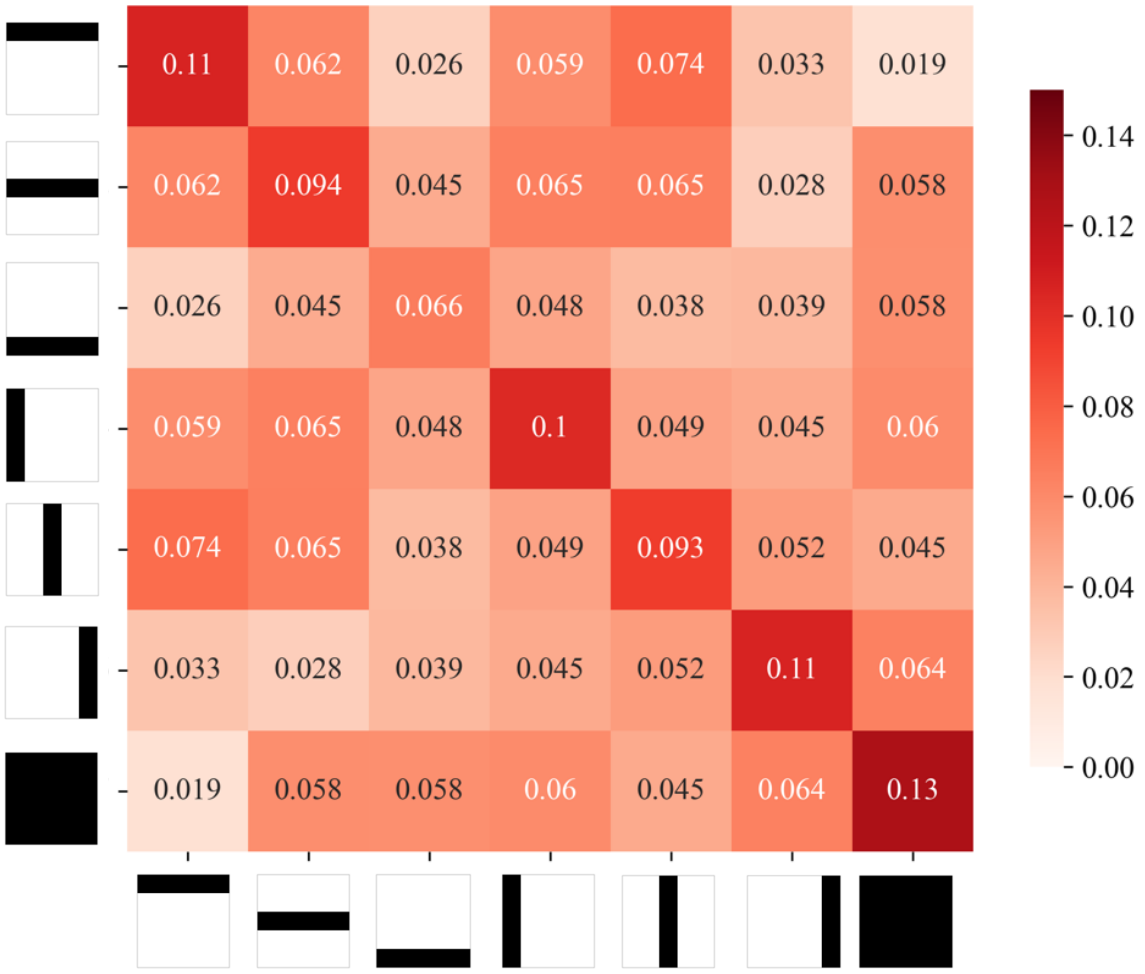

Fig. 4.34. Average correlation of generated representations by the Learner according to applied mask by the Modulator.

4.7.5 Discussion

Instead of using the zero and one vectors for inhibition as the previous simulation, this simulation used binary vectors as partial masks allowed to influence information processing more subtly. This was shown in a multi-network architecture, where a Modulator learned an array of these partial

masks—each corresponding to different features within the input space, use these to create different features of the inputs. Then, these features were learned by the Learner to generate a wider range of representations. Essentially, binary encoding enhanced the feature-extraction properties discussed in previous chapters. For instance, usually, a set of 26 representations would be generated to represent the input space. Here, 182 total representations were generated, each representing a different feature space of the inputs. These representations are more informative as they have been generated according to different aspects of the input and are formed subcategories of the input. This was shown by the correlations between these representations, where within the same mask, the representations were more similar to each other than to representations from different masks. At the same time, some representations were similar across inputs and mask, meaning this could use in a mechanism to identify similar features between patterns. Future work could manipulate the number of y-units in the Learner (FEBAM) to examine protype formation discussed in Chapter 1. Furthermore, the Learner could be a multi-FEBAM network discussed in Chapter 2, allowing an even more powerful decorrelation property. As such, different features of a task would be generated in the representations and decorrelated, which could be used to solve the non-linear tasks as in Chapter 3 using less unsupervised layers or with less epochs or could even help learn more difficult tasks.

4. 8. Chapter IV Discussion

This chapter presented a novel binary transmission function for RNAMs. This function enabled the formation of binary attractors and the removal of spurious ones arising from complements found in bipolar encoding. Its use in a BAM network confirmed that binary encoding produced fewer spurious attractors and exhibited improved recall performance under noise. Of interest, it did so while retaining the properties of the widely used bipolar encoding. Most importantly, this binary transmission function opened the door to possible new properties, as it was shown that the zero vector could be learned and used as an inhibition mechanism.

An extensive analysis of the transmission function was provided in Section 4.2. It was demonstrated that the binary differential equation had two stable (0 and 1) and one unstable fixed point (0.5). The energy landscape revealed that the local minima were indeed the fixed points. By using a Euler approximation, the ideal parameter was chosen to guarantee conversion to these stable fixed points. The final binary function came after hard limits at 0 and 1 were added to account for the absolute maximum and minimum values of neural units. This analysis was crucial to fully understand the function's stability and convergence properties and ensure that the binary transmission function could be used in an RNAM. It is essential to emphasize that two stable attractors are critical to the recurrent dynamic in an RNAM.

To evaluate the novel transmission function in a neurodynamic environment, a comparison between binary and bipolar encoding was completed using a BAM. In section 4.4, a three-dimensional vector toy example confirmed that the complement attractors were removed in the binary BAM. Interestingly, when learning these inputs, the binary BAM only had four attractors, compared to six in the BAM using bipolar encoding. These four attractors represented the three

inputs and the (0, 0, 0) zero vector. Noteworthy, this last one appears not to be the complement of the (1,1,1) vector, as its presence can be found even when (1,1,1) is not learned.

To expand on these findings, a Montecarlo simulation was also conducted in section 4.4 using higher dimensional inputs in which the number of inputs varied. It was shown in Figure 4.14c showed that, generally, the binary transmission function drastically lowered the proportion of spurious attractors. For instance, with 10 inputs binary encoding exhibited 20% compared to around 55% for the bipolar encoding. As the number of inputs to be learned increased, the proportion of spurious attractors reached 100% spurious recall was 100% for both encoding types. Under a few inputs, binary encoding showed a high proportion of spurious attractors. However, these spurious attractors were all the zero vectors. In fact, Figure 4.16a showed that until around 15-20 inputs, the zero vector is the only spurious attractor is recalled. Meaning that, without counting the zero vector, there were no spurious attractors recalled. In contrast, bipolar encoding has a theoretical limit of a minimum of 50% proportion of spurious attractors with the presence of complement attractors. Overall, this reduced proportion of spurious recall improved the noise tolerance of the model, making the network more resistant to large distortions in input. As shown in as shown in Figure 4.15, the recall performance under noise was better binary encoding regardless of the number of inputs, although the performance gap between binary and bipolar shortened as the number of inputs increased.

Therefore, the primary advantage of employing a stable bipolar representation in neural associative memories is the accelerated learning speed due to the informative value of -1. Although this concept of learning time is not an issue from a cognitive perspective and was nonetheless solved in simulation 2, these results show that the binary encoding does not negatively impact learning other than the reduced vector space. In binary encoding, no complement inputs are stored,

an overall reduction of spurious attractors and an overall increase in recall performance under noise. Interestingly, the main spurious attractor recalled was the zero vector. It may be possible that the zero vector is a common complement for each learned pattern. This could potentially be an interesting property and future research could examine leveraging the dominance of the zero vector as a spurious attractor and use it as an advantage. For instance, it could be used as an indicator for unidentifiable (never seen) inputs or as a filter when there is not enough evidence in decision-making. However, the presence of the zero attractor was limited to only a dozen or so inputs. Future research should further examine the role between memory load (number of inputs/size of inputs) and proportion of the zero vector recalled. It is of note that the purpose of this study was not to optimize performance but to establish the new binary transmission function. As such, only a few parameter values were studied. Furthermore, a more extensive analysis in binary encoding could reveal optimal values for each model parameter for task that maximizes performance as a function of learning time.

Still, as shown in section 4.5, removing complement attractors allowed the one-shot learning of a single input without storing any spurious attractors. Future work should expand on this property to explore if more than one input could be learned in one-shot learning without catastrophic forgetting. Even so, this limited one-shot learning capacity was used to create a replay system in a multi-network architecture composed of an STM and LTM. The STM combined several one-shot learning subnetworks (BAMs), which formed a continuous loop of inputs. In this way, despite only being presented with the input set once, the STM could replay the input set to the LTM, which could, in turn, learn the entire input set. As such, the LSTM could learn the task despite receiving the inputs only once from the environment, as compared to around 60 times normally. This replay system could allow the network to deal with a continuous stream of

information, where the availability or repeatability of each input might be scarce. Additionally, reducing the need for the network to receive numerous environmental inputs is more cognitively plausible. Future work should analyze the number of subnetworks and their impact on LTM learning performance.

In section 4.6, it was shown that the zero vector can be learned and used as an inhibition mechanism. As such, it can form an intentional attractor and many inputs can be associated to it. This ability was showcased in two multi-networks architectures. In the first architecture, it was shown that a network could recall the zero vector according to the context of inputs and use it to inhibit learning of certain inputs in another network. With this inhibition mechanism, it was possible to "teach" the model to filter specific unwanted stimulus. This could be used in multiple ways, such as to speed up learning of relevant inputs as not all "seen" inputs will be learned, filter noisy or random inputs, or purposely inhibit specific unwanted inputs. Moreover, the instruction set could be changed, allowing to adjust which inputs were inhibited. For instance, when changing to instruction set 2, inputs with context "2" were now learned by the Learner. Interestingly, inputs with context "1" were unlearned by the Learner network, as the error gradually increased. In other words, the Learner forgot the inputs with context 1. This is an interesting property and could be used to reset a network or remove unwanted learning. This could also help to produce more effective learning of important inputs by, for example, reducing the number of spurious attractors (Endress & Johnson, 2021). It could be very useful in situations where the instructions of a task change or more information is available which changes the answer of a task. For instance, the number of planets might have been learned as 9, but when Pluto was removed this changed to 8. Normally, in such a situation, a network would have to be retrained to learn all planets again. Here, the association of the input "Pluto" could simply be removed by inhibition. In the second

architecture, the inhibition mechanism was used by the Modulator to control what was learned in two networks depending on the context. This was also done by context, where the Modulator recalled the right decision vector which was split in half to be used on different Learners.

In future work, the inhibition properties or masking and gating could be expanded to larger multi-network architectures. Such architectures usually have the problem or having to manually set what information gets given to which network. This could be overcome by having this context-dependent gating mechanism, where the flow of information is controlled within the network. This could be very useful in situations where network dynamics need to change depending on the task. Future work should also examine or more cognitive tasks and examine other uses of binary masks, such as removing noisy parts of the input. Also, this inhibition property was expanded upon in section 4.7 by using binary vectors instead of the one or zero vectors. It was shown how masks could create features of each input, and how this could enable an unsupervised network like the FEBAM to generate representations of these features. As such, the number of features representing the input increased in proportion to the number of masks. In this case, it went from 26 to 182 for the 26 inputs and seven masks. This could be seen as allowing the FEBAM to generate a better feature set, where inputs could be compared by subcategories and not just by a representation of the overall vector. As previously discussed, this enhanced the feature-extracting properties of the FEBAM and could be combined with properties found in previous chapter. Future work should also examine how the prototype-exemplar learning, decorrelation and non-linear learning properties are enhanced by binary encoding.

In each simulation, it was the context pattern and the instruction set which determined if a response should be given or not (Rolon-Mérette, Rolon-Merette, & Chartier, 2023). However, future research could examine other methods for the Modulator to learn or recall the proper

decision vector. Furthermore, a limit is that the decision vectors, such as the zero vector, must but the same dimension as the inputs. Although this is a general problem of ANNs, future work could examine a way to overcome this limitation or by leveraging current methods made possible by using binary encoding, such as padding as the zero is not informative and will not affect learning. Importantly, the proposed architectures in Sections 4.5, 4.6 and 4.7 are proof-of-concepts to test the additional properties gained with the new binary transmission function. All networks were used the same internal functions and each network's weights were set at the beginning, meaning all properties were solely emergent on local communication within the architecture and not some external manipulation. As such, this study served to lay the foundational groundwork for future research using binary encoding. Future work should further explore binary representation-enabled properties, especially directly in cognitive tasks. For instance, it could also be possible to implement various architecture constraints, such as lateral inhibition, used in many competitive models like self-organizing maps (Kohonen, 1990). Such mechanisms could enable RNAMS to learn complex cognitive tasks like continual learning (Parisi et al., 2018; Van de Ven et al., 2020). Future studies could investigate the impact of the binary transmission function on different types of models, like unsupervised learning, self-supervised learning, multilayered architectures (Rolon-Mérette, Rolon-Mérette, & Chartier, 2023; Rolon-Merette et al., 2018), multi-model architectures (Liu et al., 2019) or in larger models like cognitive architectures (Martin et al., 2021; Petersen & Sporns, 2015). Finally, binary representations could be used to emulate the presence or absence of neuronal spikes and establish a bridge between RNAMs and Spiking Neural Networks (Lansner, 2009; Leuner et al., 2003; Stewart et al., 2011; Stöckel et al., 2017; Taherkhani et al., 2020; Wang & Cui, 2017).

4. 9. Chapter IV concluding remarks

This paper's novel binary transmission function offers a promising approach for retaining the properties of the widely used bipolar encoding while addressing its main drawbacks. The new function eliminated complement attractors, reduced spurious attractors, and improved recall performances under noise. Additionally, it offers a promising foundation for potential new cognitive properties, such as a replay and inhibition mechanism that show promise in building multi-network architectures. In the context of this chapter, the binary transmission function emerges as a favorable choice over the bipolar transmission function. Ultimately, by exploring the potential of binary-encoded BAMs, this chapter aimed to take a step closer to creating more truthful RNAMs of human cognition capable of learning complex tasks.

5. Thesis Discussion

The work in this thesis aimed to contribute to the field of psychology by offering new insights and methodologies in cognitive modelling using artificial neural networks (ANNs). Specifically, it aimed to address limitations of associative memories (AMs), a type of ANN that has already played an important role in modelling the neurodynamical processes of learning and memory. However, such networks were still limited in the complexity of associations and tasks that they could learn. As such, a central hypothesis was that the limitation of AMs could be due to their small architectures, often composed of a single network with 2 or very few layers, and that expanding this to a multi-layer and multi-network architecture would enable new properties to allow more complex tasks to be learned. Hence, the preceding chapters have built upon each other to propose gradual changes and improvements to AMs, allowing several new properties and learning of more complex tasks. A central aspect of these improvements is the consistency of network dynamics and learning rule. As such, every cognitive property discussed in this thesis is emergent from local interactions and not a direct goal of the network and not externally manipulated. As such, this discussion returns on key findings from this thesis to contextualize these contributions within the broader landscape of cognitive modelling and psychology research. Additionally, an overview of limitations, suggested directions for future research and potential avenues for applying these findings to other fields are discussed.

While each chapter of this thesis served as an independent research article, they are interconnected through a logical and constructive progression. Chapters 1 to 3 aim to improve ANNs is by modifying their architecture, i.e., changing how artificial neurons are interconnected. Specifically, Chapter 2 extends the findings of Chapter 1 to build an unsupervised multilayer architecture, Chapter 3 integrates and builds upon the insights from Chapters 1 and 2 to build an

AM capable of learning non-linear tasks. Chapter 4 creates a new transmission function to change an encoding landscape to binary to create more possible behaviours. Thus, although each section presents distinct focuses and discoveries, together they weave a coherent narrative. This structure allows for a cumulative exploration of associative memory behaviour and its applications, with each chapter contributing to a layered understanding of the subject matter, and collectively advancing the field by addressing increasingly complex questions and applications. For instance, before building a multilayer architecture of many FEBAMs, it was important to first be established if the FEBAM has a consistent behaviour.

5.1 Key chapter findings

Therefore, the first chapter examined the FEBAMs across different architectural settings by varying its number of y-units (the size of generated representations), different weight initialization and different stimuli. This investigation revealed a critical insight: despite differences in weight initialization leading to distinct representations, these representations invariably conform to the same underlying nature, either as prototypes or exemplars, consistent within the architectural confines for any given task. This uniformity in representation nature, regardless of input and weight initialization variability, suggested that architecture dictates FEBAM behaviour more than any other factor. This meant that when using many FEBAMs, the kind of processing can be predicted based on the architecture and not reliant on the random factor of weight initialization. This is crucial when building multi-network models, as knowing the properties of individual components is crucial to explain the overall behaviour of the network. A further observation was made regarding the correlation between representations; specifically, even as representations adhered to their categorical correlations, a general decline in correlation metrics was observed. This phenomenon spurred the hypothesis that employing multiple FEBAMs might lead to

complete stimulus decorrelation, a concept that transitioned into the focus of the subsequent chapter. As such, Chapter 2 explored the potential of a multi-layer FEBAM architecture. The construction of a model incorporating multiple FEBAMs revealed a progressive decorrelation of stimuli across layers, effectively separating them into a higher-dimensional space. This outcome was an emerged byproduct of the model's local, unsupervised auto-associative operations, and not the directed goal of the network. Chapter 3 built upon this premise by proposing the feasibility of transforming non-linearly separable inputs into linearly separable representations through a multi-layer FEBAM framework. The empirical findings showed that such a transformation was indeed possible, enabling a BAM to learn non-linear tasks by leveraging linearly separable representations generated by the multilayered architecture. Once again, this capability emerged not as a deliberate design objective but as a property of the model's local learning associations. The chapter also explored learning more and more complex non-linear inputs through the strategic addition of layers and units.

However, these yielded challenges of controlling information flow within the multilayered architecture, as the representations of the last layer, regardless of architecture size, was always used, which would not always be wanted or needed. As such, in an effort to implement modulatory processes to mimic human cognitive functions, a shift towards binary encoding was proposed as a solution in Chapter 4. Until now, the encoding was done under a bipolar transmission function, which is highly limited in achieving effective information control, as the both the -1 and 1 attractors are informative. A shift to binary encoding would enable using the zero as an attractor to represent lack of information, allowing to use the absorbent property of the zero during matrix multiplication. As a result, the goal of the chapter was to create and examine a binary transmission function which retained the learning properties examined with bipolar encoding. The chapter first

detailed the new binary transmission function to describe its two stable attractors at 1 and 0. Then, the first set of simulations showed that learning in the BAM and FEBAM was not affected when using this binary transmission function. In fact, although it yielded longer leaning times, it lowered the presence of spurious attractors and improved recall under noise. It was noted that under low numbers of input stimuli, regardless of input dimensionality, it was possible that the zero attractor was the only spurious attractor. Subsequently, it was shown how the zero vector could be learned by a network and how this, paired with the cognitive context method shown in Chapter 1, allowed that network to selectively inhibit learning in another network.

Importantly, the shift to binary encoding enabled a mechanism for inhibition, a crucial aspect of human cognition (Friedman & Robbins, 2022). With bipolar encoding, this was technically possible, but much more difficult without any external intervention, as the specific values of each incoming input would need to be known in advance. Instead, the absorbent property of the zero allows a greater degree of control, meaning the inhibition can be, as in chapter 4, context dependant. As such, the inhibition exhibited by the Modulator in Chapter 4 was done with minimal external influence, as only the instruction set was established. Future studies could explore situations where the instruction set is created by the network. This could perhaps be possible by self-generating contexts using the compression properties of the FEBAM to generate prototypes of inputs, which could then act as the contexts, as discussed in Chapter 1. Regardless, this inhibition property makes it feasible to explore multi-network architectures with the new binary transmission function (Hersche et al., 2023; Małkiński & Mańdziuk, 2022).

Furthermore, the inhibition property could be used as partial masks, and allowed inputs to be separated in subcategories, which could then yield more generated representations by the FEBAM. As discussed in Chapter 1 and 2, the representations generated by the FEBAM transform

the input while preserving the categories within the input. Such binary partial masks could be used in future research to enhance the FEBAMs feature-generating capabilities. For instance, the FEBAM could learn to recognize how inputs are similar or different according to their sub-features. This could be a mechanism to create an associative memory capable of learning visual stimuli. This is one of the major limitations of the AMs, as they are not designed to handle visual stimuli. While binary/bipolar vectors are represented in grid format for visual purposes, they are not inherently visual stimuli. Tasks such as the MNIST are quite difficult for RNAMs because they have a strong visual cue. A type of ANN which has shown to deal well with visual inputs are Convolution Neural Networks (CNNs). These multilayered networks use convolutional operations on visual inputs to transform them into sub-feature representations which can then be learned by standard ANNs (Lindsay, 2021). Because of the unsupervised properties of the FEBAM, like the compression properties shown in Chapter 1, the decorrelation properties of multiple layers in Chapter 2 and the partial masking in Chapter 4, it may be feasible to adapt the CNNs method and create a convolutional MF-BAM to enable learning visual stimuli.

It was also shown in Chapter 4 that one-shot learning was possible with the binary transmission function, and that this could serve as a memory replay system in a model with a short-term-memory (STM) and long-term memory (LTM). In this multi-network architecture, the STM learned a set of inputs with several one-shot learning subnetworks which could only hold one pattern into memory each. However, with a cycling of information, the STM could retain into memory the set of inputs and replay it to the LTM, which could then consolidate the entire set. As such, the LTM learned the task when the environment had presented the input only once. Such replay systems are believed to be crucial to human memory consolidation and could be key to

allowing complex behaviour such as rule learning or thinking (Jensen et al., 2024; Kaefer et al., 2020; O'Reilly et al., 2022; Peyrache et al., 2009).

5.2 A binary MF-BAM

The MF-BAM architecture presented in Chapter 3 (built upon findings from Chapters 1 and 2), was a significant step towards solving more complex tasks as it enabled the learning of non-linear tasks. This was problematic as the current methods were not cognitively plausible or required external intervention. Despite this, the model could be significantly improved by introducing binary encoding in future work. As shown in Chapter 4, the inhibition mechanism can be used as a gating mechanism, to direct the flow of information when multiple networks are used. Such gating mechanisms could also be used on the multilayer architectures developed in Chapters 2 and 3 (Rolon-Mérette, Rolon-Mérette, & Chartier, 2023; Rolon-Merette et al., 2018). It was shown that layers would gradually decorrelate inputs and linearly separate nonlinear inputs across its different layers. Yet, the desired behaviour, whether a decorrelation level or a specific decision zone, would need to be determined in advance to decide on the number of layers needed. This was problematic for three reasons. First, all layers would be trained at once, meaning it was favourable to choose an architecture with many layers which took a long time to complete learning, sometimes almost 25000 epochs. Second, it may be that a large architecture is not needed or not desired. Third, the desired behaviour could change. These limitations can be overcome by using binary encoding. For instance, by taking the MF-BAM architecture form Chapter 3 with the Modulator's inhibition method from Chapter 4, a multi-network model where a modulator could be devised to control the flow of information in the MF-BAM. An example of such an architecture is proposed in Figure 5.1.

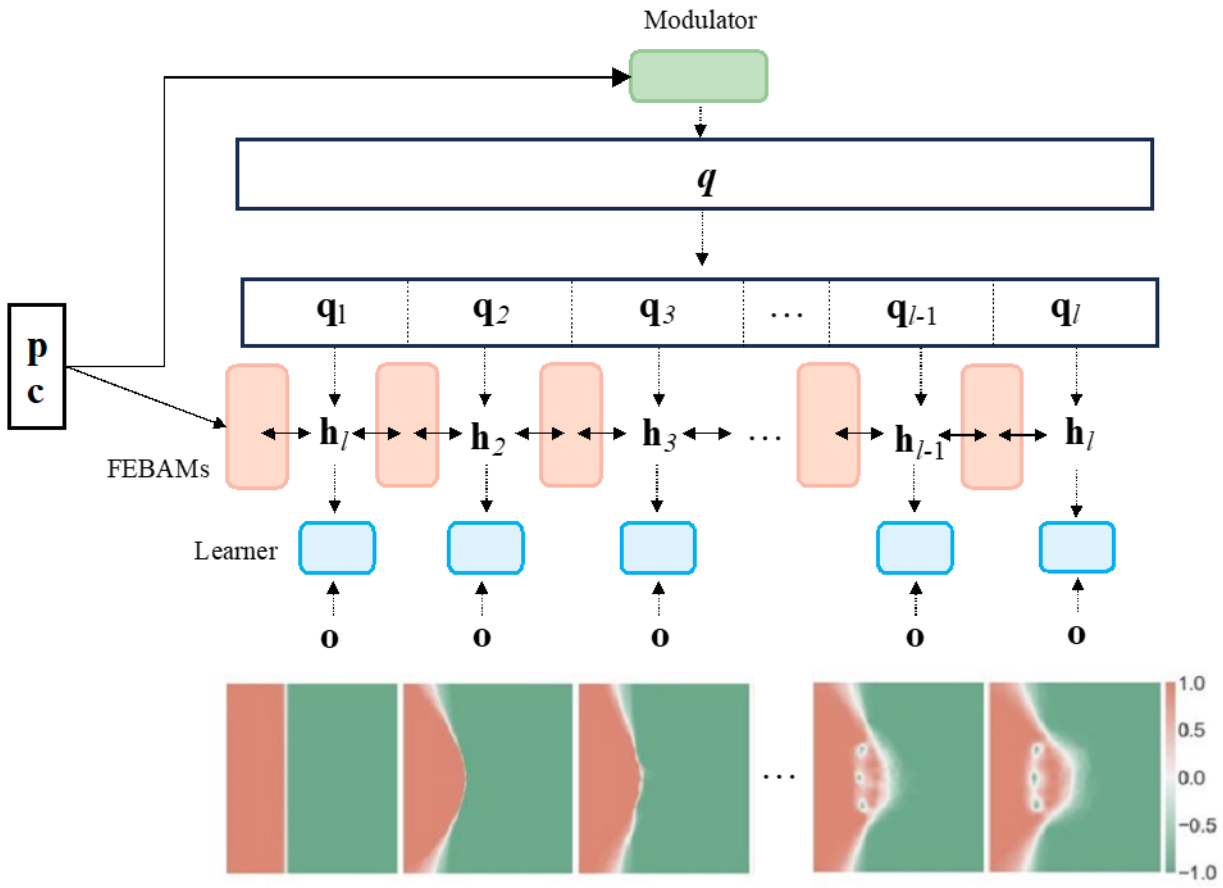

Fig. 5.1. A proposed masking/inhibition architecture in the MF-BAM

In this architecture, there is the MF-BAM and a modulator. The modulator would first learn an instruction set to associate contexts representing digits 1 to *l* to different decision vectors **q**, which could then be divided into subsets q1 to ql, similar to what is described in section 4.6. Each subset would be either a vector of zeros or ones depending on the context. These would be used on the corresponding generated representations of every layer in a Hadamard product ($\mathbf{q}_l \odot \mathbf{h}_l$). As such, the generated representations at each layer could be inhibited and transformed to a zero vector, rendering representations of all subsequent layers to also be zero vectors. Effectively, by doing so, there is a control on which generated representation will be used by the BAM (i.e. the Learner) to generate the decision boundary. For example, Figure 5.1 shows the decision boundaries at different layers for the Suns tasks from Chapter 3. If an input with context 2 is presented, then $\mathbf{q}_2$ is recalled

by the modulator and the generated representation of the third FEBAM, $\mathbf{h}_3$, will be inhibited. Then the Learner (BAM or MF) uses $\mathbf{h}_2$ to learn the task targets (**o**) and the resulting linear decision boundary is slightly non-linear. Similarly, if there is an input with context is *l*, then $\mathbf{q}_l$ is recalled by the modulator and the generated representation of the final FEBAM, $\mathbf{h}_l$, is used by the Learner the linear decision boundary exhibits the learning of all the exceptions in the Suns task. Such a process could be akin to different levels of expertise in a task. While an expert may be able to explain every detail of a particular object or task, they can still describe the basic components or in a general way. Additionally, a memory replay system could be implemented in the MF-BAM using one-shot learning architectures such as discussed in Chapter 4. For complex non-linear tasks like the Suns task, the MF-BAM needed close to 25000 learning epochs to generate the proper decision boundary, which means the task had to be presented 25000 times. Future research could explore how a memory replay system could speed this process, by replaying information to key layers within the MF. Alternatively, the inhibition property could be used on a network composed of multiple MF-BAMs. By combining the generated representations of multiple MF-BAMs learning in parallel, such complex tasks could be more easily solved. Gating with inhibition could be used to strategically control the flow of information in such a network, similar to ensemble networks (Ganaie et al., 2022; Sagi & Rokach, 2018; Zhou et al., 2022).

5.3 Future work

This thesis aimed to use ANNs to explain phenomena in neurocognition by addressing the current gaps in the scientific literature. This would thereby allow for a better understanding of the functioning of the human brain. In the short term, these discoveries could be integrated with more biologically plausible ANNs like spiking neural networks (Ghosh-Dastidar & Adeli, 2009; Yamazaki et al., 2022). Binary encoding is a step closer to biological spikes and future work could create a spiking version of the BAM based on the same internal functions used in this thesis. This could create a bridge between biological processes and cognitive functioning. On the flip side, the shift to larger architectures done throughout the chapters could allow to create or be integrated into higher cognitive models, like cognitive architectures, which could help tackle more complex tasks (Kotseruba & Tsotsos, 2020; Petersen & Sporns, 2015). It would be intriguing to explore tasks traditionally used in cognitive modelling to examine brain processes. In fact, because of the cognitive properties maintained in this thesis, such as the behaviour being a byproduct of local association as in the brain, a concrete link could be made with cognitive theories (Guest & Martin, 2023). For instance, tasks like the N-back or the Digit Span task, well-established paradigms for assessing working memory, could be an ideal starting point for modelling cognitive load (Greenberg & Zheng, 2022; Sweller, 2011). The Digit Span task involves recalling a sequence of numbers in both forward and backward orders, challenging the network's sequential memory capabilities and flexibility. The N-back task requires participants to continuously update and manipulate information in their minds while monitoring a sequence. Implementing such tasks in a neural dynamic context could help reveal how different architectural modifications influence the model’s capacity to handle increasing cognitive loads.

Furthermore, complementary learning systems theory stipulates that networks with different architectures interact together in the brain to learn rules and complete other cognitive

functions (Kumaran et al., 2016). Rule learning and continual learning are abilities at the core of human cognition allowing not only to grasp present structures and dynamics but also to transfer knowledge from one domain to another (Hassabis et al., 2017; Parisi et al., 2019; Thabtah & Peebles, 2020). Current ANN approaches use such multi-network architectures to solve tasks like the Siegler balance, a problem with non-visual stimuli requiring understanding of the weight-distance relationship in different ways (Siegler & Chen, 2002) or Raven's Progressive Matrices (Raven, 2003), analogy problems where a matrix of geometric figures is presented with a missing entry. Still, these advances still struggle with proper rule learning, as well as suffering from catastrophic forgetting. Moreover, such methods often use networks with divergent internal dynamics, which do not match the functioning of biological neurons (Hersche et al., 2023; Małkiński & Mańdziuk, 2022). Although the human brain is made up of networks with varied functions, all these networks operate with similar neurodynamics (Harnad, 1990; O'Reilly, 1998). Thus, for such ANNs to provide a plausible explanation for complex cognitive phenomena like rule learning in humans, the different interacting networks must operate by having similar internal functions. This thesis takes a significant step towards the use multiple ANN interactions to achieve complex cognitive properties like rule learning.

Although many multi-network architectures shown in this thesis are still proof of concepts, the properties gained with binary encoding, along with the architectural exploration done in this thesis chapters, provide a framework for the flexible development of new architectures. Importantly, such architectures, whether multilayered or multi-network, will retain the same internal functioning throughout its components and any behaviour will be the result of local associative learning. Thus, along with the Siegler balance or the Ravens progressive matrices, extending to other tasks commonly used in psychology could be possible. Tasks such as the Tower

of Hanoi or the Wisconsin Card Sorting Task (WCST) require rule learning (Kotovsky et al., 1985; Test, 1993). The Tower of Hanoi, a problem-solving task requiring participants to move a stack of discs from one peg to another following specific rules, assesses strategic planning and sequential decision-making. In the WCST, participants must shift between different sorting rules based on feedback. By modeling these tasks, ANNs could help elucidate how the brain learns and applies abstract rules and adapts to changing conditions, further enhancing our understanding of cognitive control and flexibility. Moreover, natural language tasks, such as sentence comprehension or text generation, could be interesting. It would be of particular interest to examine how language could be learned without the need for an overwhelming amount of data. Exploring such tasks could help understand how complex relationships between stimuli and responses are formed and modified. Thus, bridging the gap between specific task performance and general associative mechanisms observed in the brain.

6. Conclusion

Ultimately, the work in this thesis hopes to form a significant step forward in bridging artificial neural networks (ANNs) with cognitive models to enhance our understanding of brain function and complex cognitive tasks. Through multilayered and multi-network frameworks that retain consistent internal dynamics, this research offers cognitively and biologically plausible approaches to solving non-linear tasks, controlling information flow, and facilitating properties like inhibition and memory replay. Overall, systematically developing and refining architectures like Associative Memories with a balance between plausibility and computational proficiency can help ANNs improve performance, achieve new properties and increase explanatory power. Importantly, these findings emphasize that complex cognitive properties such as learning, memory consolidation, and inhibition can emerge from local neural interactions in the dynamical systems of large neural networks. Future research could build on these insights by integrating these models with more biologically plausible networks or applying them to real-world cognitive architectures. Overall, this thesis lays the groundwork for a more nuanced and holistic approach to understanding human cognition through artificial intelligence, highlighting the potential of ANNs not just as computational tools but as meaningful cognitive models.

References

Abbott, L. (2006). Where are the switches on this thing. *23 Problems in systems neuroscience*, 423-431.

Acevedo-Mosqueda, M. E., Yáñez-Márquez, C., & Acevedo-Mosqueda, M. A. (2013). Bidirectional associative memories: Different approaches. *ACM Computing Surveys (CSUR)*, *45*(2), 1-30.

Acevedo-Mosqueda, M. E., Yáñez-Márquez, C., & López-Yáñez, I. (2006). A new model of BAM: Alpha-beta bidirectional associative memories. Computer and Information Sciences–ISCIS 2006: 21th International Symposium, Istanbul, Turkey, November 1-3, 2006. Proceedings 21,

Ackman, J. B., & Crair, M. C. (2014). Role of emergent neural activity in visual map development. *Current opinion in neurobiology*, *24*, 166-175.

Adigun, O., & Kosko, B. (2019). Bidirectional backpropagation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *50*(5), 1982-1994.

Aggarwal, C. C. (2018). *Neural networks and deep learning* (Vol. 10). Springer.

Ahmed, M. S., Priestley, J. B., Castro, A., Stefanini, F., Canales, A. S. S., Balough, E. M., Lavoie, E., Mazzucato, L., Fusi, S., & Losonczy, A. (2020). Hippocampal network reorganization underlies the formation of a temporal association memory. *Neuron*, *107*(2), 283-291. e286.

Alamia, A., Gauducheau, V., Paisios, D., & VanRullen, R. (2020). Comparing feedforward and recurrent neural network architectures with human behavior in artificial grammar learning. *Scientific reports*, *10*(1), 22172.

Amit, D. J., & Amit, D. J. (1989). *Modeling brain function: The world of attractor neural networks*. Cambridge university press.

Anderson, J. A. (1983). Cognitive and psychological computation with neural models. *IEEE transactions on systems, man, and cybernetics*(5), 799-815.

Anderson, J. A., Silverstein, J. W., Ritz, S. A., & Jones, R. S. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological review*, *84*(5), 413.

Anderson, J. R., & Bower, G. H. (1972). Recognition and retrieval processes in free recall. *Psychological review*, *79*(2), 97.

Anderson, J. R., & Bower, G. H. (2014). *Human associative memory*. Psychology press.

Arthur, D., & Vassilvitskii, S. (2006). *k-means++: The advantages of careful seeding*.

Bank, D., Koenigstein, N., & Giryes, R. (2023). Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, 353-374.

Barak, O. (2017). Recurrent neural networks as versatile tools of neuroscience research. *Current opinion in neurobiology*, *46*, 1-6.

Baroni, T. J. (2017). *Mushrooms of the northeastern United States and eastern Canada*. Timber Press.

Barra, A., Catania, G., Decelle, A., & Seoane, B. (2023). Thermodynamics of bidirectional associative memories. *Journal of Physics A: Mathematical and Theoretical*, *56*(20), 205005.

Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, *43*(1), 3-31.

Beniaguev, D., Segev, I., & London, M. (2021). Single cortical neurons as deep artificial neural networks. *Neuron*, *109*(17), 2727-2739. e2723.

Beyeler, M., Rounds, E. L., Carlson, K. D., Dutt, N., & Krichmar, J. L. (2019). Neural correlates of sparse coding and dimensionality reduction. *PLoS computational biology*, *15*(6), e1006908.

Blakeman, S., & Mareschal, D. (2020). A complementary learning systems approach to temporal difference learning. *Neural Networks*, *122*, 218-230.

Blazek, P. J., & Lin, M. M. (2021). Explainable neural networks that simulate reasoning. *Nature Computational Science*, *1*(9), 607-618.

Brendel, W., Bourdoukan, R., Vertechi, P., Machens, C. K., & Denéve, S. (2020). Learning to represent signals spike by spike. *PLoS computational biology*, *16*(3), e1007692.

Buckner, R. L., & DiNicola, L. M. (2019). The brain's default network: updated anatomy, physiology and evolving insights. *Nature Reviews Neuroscience*, *20*(10), 593-608.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., & Grobler, J. (2013). API design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*.

Carpenter, G. A., & Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, *21*(3), 77-88.

Caudill, M., & Butler, C. T. (1990). *Naturally intelligent systems*. MIT press.

Chartier, S., & Boukadoum, M. (2006a). A bidirectional heteroassociative memory for binary and grey-level patterns. *IEEE Transactions on neural networks*, *17*(2), 385-396.

Chartier, S., & Boukadoum, M. (2006b). A sequential dynamic heteroassociative memory for multistep pattern recognition and one-to-many association. *IEEE Transactions on neural networks*, *17*(1), 59-68.

Chartier, S., & Boukadoum, M. (2011). Encoding static and temporal patterns with a bidirectional heteroassociative memory. *Journal of applied mathematics*, *2011*(1), 301204.

Chartier, S., Boukadoum, M., & Amiri, M. (2009). BAM learning of nonlinearly separable tasks by using an asymmetrical output function and reinforcement learning. *IEEE Transactions on neural networks*, *20*(8), 1281-1292.

Chartier, S., Giguère, G., Renaud, P., Lina, J.-M., & Proulx, R. (2007). FEBAM: A feature-extracting bidirectional associative memory. 2007 International Joint Conference on Neural Networks,

Chartier, S., Leth-Steensen, C., & Hebert, M.-F. (2012). Performing complex associations using a generalised bidirectional associative memory. *Journal of Experimental & Theoretical Artificial Intelligence*, *24*(1), 23-42.

Chartier, S., & Proulx, R. (2005). NDRAM: Nonlinear dynamic recurrent associative memory for learning bipolar and nonbipolar correlated patterns. *IEEE Transactions on neural networks*, *16*(6), 1393-1400.

Chauhan, V. K., Dahiya, K., & Sharma, A. (2019). Problem formulations and solvers in linear SVM: a review. *Artificial Intelligence Review*, *52*, 803-855.

Christophel, T. B., Klink, P. C., Spitzer, B., Roelfsema, P. R., & Haynes, J.-D. (2017). The distributed nature of working memory. *Trends in cognitive sciences*, *21*(2), 111-124.

Church, K., Rolon-Mérette, T., Ross, M., & Rolon-Mérette, D. (2021). Introduction to Python's Syntax. *The Quantitative Methods for Psychology*, *17*(1).

Church, K., Ross, M., & Chartier, S. (2020). Using a bidirectional associative memory and feature extraction to model nonlinear exploitation problems. Proceedings of the 18th International Conference on Cognitive Modelling,

Churchland, M. M., Yu, B. M., Cunningham, J. P., Sugrue, L. P., Cohen, M. R., Corrado, G. S., Newsome, W. T., Clark, A. M., Hosseini, P., & Scott, B. B. (2010). Stimulus onset quenches neural variability: a widespread cortical phenomenon. *Nature neuroscience*, *13*(3), 369-378.

Cichy, R. M., & Kaiser, D. (2019). Deep neural networks as scientific models. *Trends in cognitive sciences*, *23*(4), 305-317.

Díaz de León, J. L., & Gamino Carranza, A. (2022). New binary associative memory model based on the XOR operation. *Applicable Algebra in Engineering, Communication and Computing*, *33*(3), 283-320.

Eliasmith, C. (2013). *How to build a brain: A neural architecture for biological cognition*. OUP USA.

Elman, J. L. (1996). *Rethinking innateness: A connectionist perspective on development* (Vol. 10). MIT press.

Endress, A. D., & Johnson, S. P. (2021). When forgetting fosters learning: A neural network model for statistical learning. *Cognition*, *213*, 104621.

Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., & Prager, J. (2010). Building Watson: An overview of the DeepQA project. *AI magazine*, *31*(3), 59-79.

Fiori, S. (2000). An experimental comparison of three PCA neural networks. *Neural Processing Letters*, *11*, 209-218.

Földiak, P. (1990). Forming sparse representations by local anti-Hebbian learning. *Biological cybernetics*, *64*(2), 165-170.

Forstmann, B. U., Wagenmakers, E.-J., Eichele, T., Brown, S., & Serences, J. T. (2011). Reciprocal relations between cognitive neuroscience and formal cognitive models: opposites attract? *Trends in cognitive sciences*, *15*(6), 272-279.

Friedman, N. P., & Robbins, T. W. (2022). The role of prefrontal cortex in cognitive control and executive function. *Neuropsychopharmacology*, *47*(1), 72-89.

Ganaie, M. A., Hu, M., Malik, A. K., Tanveer, M., & Suganthan, P. N. (2022). Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, *115*, 105151.

Genon, S., Eickhoff, S. B., & Kharabian, S. (2022). Linking interindividual variability in brain structure to behaviour. *Nature Reviews Neuroscience*, *23*(5), 307-318.

Ghosh-Dastidar, S., & Adeli, H. (2009). Spiking neural networks. *International journal of neural systems*, *19*(04), 295-308.

Giguère, G., Chartier, S., Proulx, R., & Lina, J.-M. (2007). Category development and reorganization using a bidirectional associative memory-inspired architecture. Proceedings of the 8th international conference on cognitive modeling,

Goldstein, E. B. (2015). *Cognitive psychology: Connecting mind, research and everyday experience*. Cengage Learning Stamford, CT.

Goodwin, G. P., & Johnson-Laird, P. N. (2013). The acquisition of Boolean concepts. *Trends in cognitive sciences*, *17*(3), 128-133.

Greenberg, K., & Zheng, R. (2022). Cognitive load theory and its measurement: a study of secondary tasks in relation to working memory. *Journal of Cognitive Psychology*, *34*(4), 497-515.

Grewe, B. F., Gründemann, J., Kitch, L. J., Lecoq, J. A., Parker, J. G., Marshall, J. D., Larkin, M. C., Jercog, P. E., Grenier, F., & Li, J. Z. (2017). Neural ensemble dynamics underlying a long-term associative memory. *Nature*, *543*(7647), 670-675.

Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, *11*(1), 23-63.

Grossberg, S. (2013). Adaptive Resonance Theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural Networks*, *37*, 1-47.

Guest, O., & Martin, A. E. (2023). On logical inference over brains, behaviour, and artificial neural networks. *Computational Brain & Behavior*, *6*(2), 213-227.

Hampton, J. A. (1997). Associative and similarity-based processes in categorization decisions. *Memory & cognition*, *25*, 625-640.

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, *42*(1-3), 335-346.

Hassabis, D., Kumaran, D., Summerfield, C., & Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, *95*(2), 245-258.

Hasson, U., Nastase, S. A., & Goldstein, A. (2020). Direct fit to nature: an evolutionary perspective on biological and artificial neural networks. *Neuron*, *105*(3), 416-434.

Hayes, T. L., Krishnan, G. P., Bazhenov, M., Siegelmann, H. T., Sejnowski, T. J., & Kanan, C. (2021). Replay in deep learning: Current approaches and missing biological elements. *Neural computation*, *33*(11), 2908-2950.

Haykin, S. (1998). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Haykin, S. (2009). *Neural networks and learning machines, 3/E*. Pearson Education India.

He, H., Shang, Y., Yang, X., Di, Y., Lin, J., Zhu, Y., Zheng, W., Zhao, J., Ji, M., & Dong, L. (2019). Constructing an associative memory system using spiking neural network. *Frontiers in Neuroscience*, *13*, 650.

Hebb, D., O. (1949). The organization of behavior. *J Clin Psychol. new york: Wiley*, *6*(3), 335-307.

Hersche, M., Zeqiri, M., Benini, L., Sebastian, A., & Rahimi, A. (2023). A neuro-vector-symbolic architecture for solving Raven's progressive matrices. *Nature Machine Intelligence*, *5*(4), 363-375.

Hesselmann, G., Kell, C. A., Eger, E., & Kleinschmidt, A. (2008). Spontaneous local variations in ongoing neural activity bias perceptual decisions. *Proceedings of the National Academy of Sciences*, *105*(31), 10984-10989.

Hintzman, D. L. (2014). Why are formal models useful in psychology? In *Relating theory and data* (pp. 39-56). Psychology Press.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., & Peste, A. (2021). Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, *22*(241), 1-124.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554-2558. https://doi.org/doi:10.1073/pnas.79.8.2554

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., & Guadarrama, S. (2017). Speed/accuracy trade-offs for modern convolutional object

detectors. Proceedings of the IEEE conference on computer vision and pattern recognition,

Inc., A. (2020). Anaconda software distribution. *Computer software. Anaconda Documentation*.

Jain, A. K., Mao, J., & Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *Computer*, *29*(3), 31-44.

Jensen, K. T., Hennequin, G., & Mattar, M. G. (2024). A recurrent network model of planning explains hippocampal replay and human behavior. *Nature neuroscience*, 1-9.

Kaefer, K., Nardin, M., Blahna, K., & Csicsvari, J. (2020). Replay of behavioral sequences in the medial prefrontal cortex during rule switching. *Neuron*, *106*(1), 154-165. e156.

Kaefer, K., Stella, F., McNaughton, B. L., & Battaglia, F. P. (2022). Replay, the default mode network and the cascaded memory systems model. *Nature Reviews Neuroscience*, *23*(10), 628-640.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, *24*(7), 881-892.

Kemker, R., McClure, M., Abitino, A., Hayes, T., & Kanan, C. (2018). Measuring catastrophic forgetting in neural networks. Proceedings of the AAAI conference on artificial intelligence,

Knoblauch, A. (2005). Neural associative memory for brain modeling and information retrieval. *Information Processing Letters*, *95*(6), 537-544.

Kodinariya, T. M., & Makwana, P. R. (2013). Review on determining number of Cluster in K-Means Clustering. *International Journal*, *1*(6), 90-95.

Kohonen, T. (1974). An adaptive associative memory principle. *IEEE transactions on computers*, *100*(4), 444-445.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, *78*(9), 1464-1480.

Kohonen, T., Lehtiö, P., Rovamo, J., Hyvärinen, J., Bry, K., & Vainio, L. (1977). A principle of neural associative memory. *Neuroscience*, *2*(6), 1065-1076.

Kosko, B. (1988). Bidirectional associative memories. *IEEE transactions on systems, man, and cybernetics*, *18*(1), 49-60.

Kosko, B. (2021). Bidirectional associative memories: unsupervised Hebbian learning to bidirectional backpropagation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *51*(1), 103-115.

Kotovsky, K., Hayes, J. R., & Simon, H. A. (1985). Why are some problems hard? Evidence from Tower of Hanoi. *Cognitive psychology*, *17*(2), 248-294.

Kotseruba, I., & Tsotsos, J. K. (2020). 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, *53*(1), 17-94.

Kriete, T. E., & Noelle, D. C. (2005). Models of Cognition: Neurological Possiblity Does Not Indicate Neurological Plausibility. Proceedings of the Annual Meeting of the Cognitive Science Society,

Kumar, C. A., Ishwarya, M., & Loo, C. K. (2015). Formal concept analysis approach to cognitive functionalities of bidirectional associative memory. *Biologically Inspired Cognitive Architectures*, *12*, 20-33.

Kumaran, D., Hassabis, D., & McClelland, J. L. (2016). What learning systems do intelligent agents need? Complementary learning systems theory updated. *Trends in cognitive sciences*, *20*(7), 512-534.

Labib, R. (1999). New single neuron structure for solving nonlinear problems. IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339),

Lansner, A. (2009). Associative memory models: from the cell-assembly theory to biophysically detailed cortex simulations. *Trends in neurosciences*, *32*(3), 178-186.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436-444.

Lee, S. W., O'Doherty, J. P., & Shimojo, S. (2015). Neural computations mediating one-shot learning in the human brain. *PLoS biology*, *13*(4), e1002137.

Leijnen, S., & Veen, F. v. (2020). The neural network zoo. Proceedings,

Letzkus, J. J., Wolff, S. B., & Lüthi, A. (2015). Disinhibition, a circuit mechanism for associative learning and memory. *Neuron*, *88*(2), 264-276.

Leuner, B., Falduto, J., & Shors, T. J. (2003). Associative memory formation increases the observation of dendritic spines in the hippocampus. *Journal of Neuroscience*, *23*(2), 659-665.

Levering, K. R., Conaway, N., & Kurtz, K. J. (2020). Revisiting the linear separability constraint: New implications for theories of human category learning. *Memory & cognition*, *48*, 335-347.

Li, X., Zhou, Y., Pan, Z., & Feng, J. (2019). Partial order pruning: for best speed/accuracy trade-off in neural architecture search. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition,

Li, Y., Li, J., Li, J., Duan, S., Wang, L., & Guo, M. (2021). A reconfigurable bidirectional associative memory network with memristor bridge. *Neurocomputing*, *454*, 382-391.

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., & Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, *21*(6), 335-346.

Lim, S. (2021). Hebbian learning revisited and its inference underlying cognitive function. *Current Opinion in Behavioral Sciences, 38*, 96-102.

Lindsay, G. W. (2021). Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of cognitive neuroscience*, *33*(10), 2017-2031.

Liu, J., Gong, M., & He, H. (2019). Deep associative neural network for associative memory based on unsupervised representation learning. *Neural Networks, 113*, 41-53.

Liu, S., Ni'mah, I., Menkovski, V., Mocanu, D. C., & Pechenizkiy, M. (2021). Efficient and effective training of sparse recurrent neural networks. *Neural Computing and Applications, 33*, 9625-9636.

Macpherson, T., Churchland, A., Sejnowski, T., DiCarlo, J., Kamitani, Y., Takahashi, H., & Hikida, T. (2021). Natural and Artificial Intelligence: A brief introduction to the interplay between AI and neuroscience research. *Neural Networks, 144*, 603-613.

Małkiński, M., & Mańdziuk, J. (2022). Deep learning methods for abstract visual reasoning: A survey on raven's progressive matrices. *arXiv preprint arXiv:2201.12382*.

Marcus, G. (2018). Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*.

Mareschal, D., French, R. M., & Quinn, P. C. (2000). A connectionist account of asymmetric category learning in early infancy. *Developmental psychology*, *36*(5), 635.

Markram, H., Muller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., Ailamaki, A., Alonso-Nanclares, L., Antille, N., & Arsever, S. (2015). Reconstruction and simulation of neocortical microcircuitry. *Cell*, *163*(2), 456-492.

Martin, L., Jaime, K., Ramos, F., & Robles, F. (2021). Declarative working memory: A bio-inspired cognitive architecture proposal. *Cognitive Systems Research, 66*, 30-45.

Masse, N. Y., Grant, G. D., & Freedman, D. J. (2018). Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, *115*(44), E10467-E10475.

Mayes, A., Montaldi, D., & Migo, E. (2007). Associative memory and the medial temporal lobes. *Trends in cognitive sciences*, *11*(3), 126-135.

Mazzoni, P., Andersen, R. A., & Jordan, M. I. (1991). A more biologically plausible learning rule for neural networks. *Proceedings of the National Academy of Sciences*, *88*(10), 4433-4437.

McClelland, J. L. (2000). Connectionist models of memory. *The Oxford handbook of memory*, 583-596.

McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, *102*(3), 419.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*, 115-133.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, *63*(2), 81.

Morissette, L., & Chartier, S. (2013). FEBAMSOM-BAM*: Neural network model of human categorization of the N-bits parity problem. The 2013 International Joint Conference on Neural Networks (IJCNN),

Mueller, S., Wang, D., Fox, M. D., Yeo, B. T., Sepulcre, J., Sabuncu, M. R., Shafee, R., Lu, J., & Liu, H. (2013). Individual variability in functional connectivity architecture of the human brain. *Neuron*, *77*(3), 586-595.

Nadler, R., Lin, L., & Minda, J. (2008). The effect of regulatory fit on the learning of complex rule-based categories. *Canadian Journal of Experimental Psychology*, *62*(4), 285.

Nikouei Mahani, M.-A., Haghgoo, H. A., Azizi, S., & Nili Ahmadabadi, M. (2016). Attention cueing and activity equally reduce false alarm rate in visual-auditory associative learning through improving memory. *Plos one*, *11*(6), e0157680.

O'Reilly, R. C. (1996). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation*, *8*(5), 895-938.

O'Reilly, R. C. (1998). Six principles for biologically based computational models of cortical cognition. *Trends in cognitive sciences*, *2*(11), 455-462.

O'reilly, R. C., & Munakata, Y. (2000). *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain*. MIT press.

O'Reilly, R. C., & Norman, K. A. (2002). Hippocampal and neocortical contributions to memory: Advances in the complementary learning systems framework. *Trends in cognitive sciences*, *6*(12), 505-510.

O'Reilly, R. C., Bhattacharyya, R., Howard, M. D., & Ketz, N. (2014). Complementary learning systems. *Cognitive science*, *38*(6), 1229-1248.

O'Reilly, R. C., Ranganath, C., & Russin, J. L. (2022). The structure of systematicity in the brain. *Current directions in psychological science*, *31*(2), 124-130.

Ólafsdóttir, H. F., Bush, D., & Barry, C. (2018). The role of hippocampal replay in memory and planning. *Current biology*, *28*(1), R37-R50.

Palm, G. (1980). On associative memory. *Biological cybernetics*, *36*(1), 19-31.

Palm, G. (2013). Neural associative memories and sparse coding. *Neural Networks*, *37*, 165-171.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, *113*, 54-71.

Parisi, G. I., Tani, J., Weber, C., & Wermter, S. (2018). Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization. *Frontiers in neurorobotics*, *12*, 78.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., & Dubourg, V. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, 2825-2830.

Perconti, P., & Plebe, A. (2020). Deep learning and cognitive science. *Cognition*, *203*, 104365.

Personnaz, L., Guyon, I., & Dreyfus, G. (1986). Collective computational properties of neural networks: New learning mechanisms. *Physical Review A*, *34*(5), 4217.

Petersen, S. E., & Sporns, O. (2015). Brain networks and cognitive architectures. *Neuron*, *88*(1), 207-219.

Peyrache, A., Khamassi, M., Benchenane, K., Wiener, S. I., & Battaglia, F. P. (2009). Replay of rule-learning related neural patterns in the prefrontal cortex during sleep. *Nature neuroscience*, *12*(7), 919-926.

Popa, C.-A. (2017). Lie algebra-valued bidirectional associative memories. Recent Advances in Soft Computing: Proceedings of the 22nd International Conference on Soft Computing (MENDEL 2016) held in Brno, Czech Republic, at June 8-10, 2016 22,

Puig, M. V., Antzoulatos, E. G., & Miller, E. K. (2014). Prefrontal dopamine in associative learning and memory. *Neuroscience*, *282*, 217-229.

Pulvermüller, F., Tomasello, R., Henningsen-Schomers, M. R., & Wennekers, T. (2021). Biological constraints on neural network models of cognitive function. *Nature Reviews Neuroscience*, *22*(8), 488-502.

Raichle, M. E. (2010). Two views of brain function. *Trends in cognitive sciences*, *14*(4), 180-190.

Raschka, S. (2018). Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*.

Raven, J. (2003). Raven progressive matrices. In *Handbook of nonverbal assessment* (pp. 223-237). Springer.

Rescorla, R. A. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and non-reinforcement. *Classical conditioning, Current research and theory*, *2*, 64-69.

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., Clopath, C., Costa, R. P., de Berker, A., & Ganguli, S. (2019). A deep learning framework for neuroscience. *Nature neuroscience*, *22*(11), 1761-1770.

Ritter, G. X., Diaz-de-Leon, J., & Sussner, P. (1999). Morphological bidirectional associative memories. *Neural Networks*, *12*(6), 851-867.

Rolon-Mérette, D., Rolon-Merette, T., & Chartier, S. (2023). Using bidirectional associative memory neural networks to solve the N-bit task. The International FLAIRS Conference Proceedings,

Rolon-Mérette, D., Rolon-Mérette, T., & Chartier, S. (2018). Distinguishing Highly Correlated Patterns using a Context Based Aproach in Bidirectional Associative Memory. 2018 International Joint Conference on Neural Networks (IJCNN),

Rolon-Mérette, D., Rolon-Mérette, T., & Chartier, S. (2019). Learning and recalling arbitrary lists of overlapping exemplars in a recurrent artificial neural network. Proceedings of the International Conference on Cognitive Modelling,

Rolon-Mérette, D., Rolon-Mérette, T., & Chartier, S. (2023). A multilayered bidirectional associative memory model for learning nonlinear tasks. *Neural Networks*, *167*, 244-265.

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T., & Church, K. (2016). Introduction to Anaconda and Python: Installation and setup. *Quant. Methods Psychol*, *16*(5), S3-S11.

Rolon-Mérette, T., Rolon-Mérette, D., Calderini, M., & Chartier, S. (2019). Different brain, same prototype? Cognitive variability within a recurrent associative memory. Proceedings of the 17th International Conference on Cognitive Modeling,

Rolon-Merette, T., Rolon-Merette, D., & Chartier, S. (2018). Generating cognitive context with feature-extracting bidirectional associative memory. *Procedia computer science*, *145*, 428-436.

Roscow, E. L., Chua, R., Costa, R. P., Jones, M. W., & Lepora, N. (2021). Learning offline: memory replay in biological and artificial reinforcement learning. *Trends in neurosciences*, *44*(10), 808-821.

Rosedahl, L. A., & Ashby, F. G. (2022). Linear separability, irrelevant variability, and categorization difficulty. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *48*(2), 159.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. (2013). Backpropagation: The basic theory. In *Backpropagation* (pp. 1-34). Psychology Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533-536.

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, *8*(4), e1249.

Saxe, A., Nelli, S., & Summerfield, C. (2021). If deep learning is the answer, what is the question? *Nature Reviews Neuroscience*, *22*(1), 55-67.

Shao, F., & Shen, Z. (2023). How can artificial neural networks approximate the brain? *Frontiers in Psychology*, *13*, 970214.

Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310-316.

Shen, Y., Wang, B., Chen, F., & Cheng, L. (2004). A new multi-output neural model with tunable activation function and its applications. *Neural Processing Letters*, *20*, 85-104.

Shi, J., & Zeng, Z. (2020). Design of in-situ learning bidirectional associative memory neural network circuit with memristor synapse. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *5*(5), 743-754.

Shields, P. J., & Rovee-Collier, C. (1992). Long-term memory for context-specific category information at six months. *Child development*, *63*(2), 245-259.

Siegler, R. S., & Chen, Z. (2002). Development of rules and strategies: Balancing the old and the new. *Journal of experimental child psychology*, *81*(4), 446-457.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., & Lanctot, M. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484-489.

Singh, M. P., & Saraswat, V. (2017). Multilayer feed forward neural networks for non-linear continuous bidirectional associative memory. *Applied Soft Computing*, *61*, 700-713.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and brain sciences*, *11*(1), 1-23.

Song, Y., Lukasiewicz, T., Xu, Z., & Bogacz, R. (2020). Can the brain do backpropagation?---exact implementation of backpropagation in predictive coding networks. *Advances in neural information processing systems*, *33*, 22566-22579.

Spooner, R. K., Wiesman, A. I., O'Neill, J., Schantell, M. D., Fox, H. S., Swindells, S., & Wilson, T. W. (2020). Prefrontal gating of sensory input differentiates cognitively impaired and unimpaired aging adults with HIV. *Brain Communications*, *2*(2), fcaa080.

Sporns, O., Tononi, G., & Kötter, R. (2005). The human connectome: a structural description of the human brain. *PLoS computational biology*, *1*(4), e42.

Stalnaker, T. A., Calhoon, G. G., Ogawa, M., Roesch, M. R., & Schoenbaum, G. (2010). Neural correlates of stimulus-response and response-outcome associations in dorsolateral versus dorsomedial striatum. *Frontiers in integrative neuroscience*, *4*, 1394.

Standage, D., Blohm, G., & Dorris, M. C. (2014). On the neural implementation of the speed-accuracy trade-off. *Frontiers in Neuroscience*, *8*, 236.

Steinbuch, K. (1961). Die lernmatrix. *Kybernetik*, *1*, 36-45.

Stewart, T. C., Tang, Y., & Eliasmith, C. (2011). A biologically realistic cleanup memory: Autoassociation in spiking neurons. *Cognitive Systems Research*, *12*(2), 84-92.

Stöckel, A., Jenzen, C., Thies, M., & Rückert, U. (2017). Binary associative memories as a benchmark for spiking neuromorphic hardware. *Frontiers in computational neuroscience*, *11*, 71.

Stork. (1989). Is backpropagation biologically plausible? International 1989 Joint Conference on Neural Networks,

Storkey, A. J., & Valabregue, R. (1999). The basins of attraction of a new Hopfield learning rule. *Neural Networks*, *12*(6), 869-876.

Südhof, T. C. (2021). The cell biology of synapse formation. *Journal of Cell Biology*, *220*(7), e202103052.

Sun, W., Advani, M., Spruston, N., Saxe, A., & Fitzgerald, J. E. (2023). Organizing memories for generalization in complementary learning systems. *Nature neuroscience*, *26*(8), 1438-1448.

Sussner, P., & Campiotti, I. (2020). Extreme learning machine for a new hybrid morphological/linear perceptron. *Neural Networks*, *123*, 288-298.

Sussner, P., & Valle, M. E. (2006). Gray-scale morphological associative memories. *IEEE Transactions on neural networks*, *17*(3), 559-570.

Suzuki, W. A. (2008). Associative learning signals in the brain. *Progress in brain research*, *169*, 305-320.

Sweller, J. (2011). Cognitive load theory. In *Psychology of learning and motivation* (Vol. 55, pp. 37-76). Elsevier.

Sweller, J., Van Merrienboer, J. J., & Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. *Educational psychology review*, *31*, 261-292.

Taherkhani, A., Belatreche, A., Li, Y., Cosma, G., Maguire, L. P., & McGinnity, T. M. (2020). A review of learning in biologically plausible spiking neural networks. *Neural Networks*, *122*, 253-272.

Tervaniemi, M., Kruck, S., De Baene, W., Schröger, E., Alter, K., & Friederici, A. D. (2009). Top-down modulation of auditory processing: effects of sound context, musical expertise and attentional focus. *European Journal of Neuroscience*, *30*(8), 1636-1642.

Test, W. C. S. (1993). Wisconsin Card Sorting Test. *Odessa, FL: Psychological Assessment Resources*.

Thabtah, F., & Peebles, D. (2020). A new machine learning model based on induction of rules for autism detection. *Health informatics journal*, *26*(1), 264-286.

Thomas, M. S., & McClelland, J. L. (2008). Connectionist models of cognition. *The Cambridge handbook of computational psychology*, 23-58.

Thompson, P. M., Schwartz, C., Lin, R. T., Khan, A. A., & Toga, A. W. (1996). Three-dimensional statistical analysis of sulcal variability in the human brain. *Journal of Neuroscience*, *16*(13), 4261-4274.

Tremblay, C., Myers-Stewart, K., Morissette, L., & Chartier, S. (2013). Bidirectional associative memory and learning of nonlinearly separable tasks. Proceedings of ICCM,

Urban, C. J., & Gates, K. M. (2021). Deep learning: A primer for psychologists. *Psychological Methods*, *26*(6), 743.

Van de Ven, G. M., Siegelmann, H. T., & Tolias, A. S. (2020). Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, *11*(1), 4069.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., & Georgiev, P. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350-354.

Wang, J.-H., & Cui, S. (2017). Associative memory cells: formation, function and perspective. *F1000Research*, *6*.

Wang, J.-H., & Cui, S. (2018). Associative memory cells and their working principle in the brain. *F1000Research*, *7*.

Wasserman, E. A., Brooks, D. I., & McMurray, B. (2015). Pigeons acquire multiple categories in parallel via associative learning: A parallel to human word learning? *Cognition*, *136*, 99-122.

Wattenmaker, W. D. (1995). Knowledge structures and linear separability: Integrating information in object and social categorization. *Cognitive psychology*, *28*(3), 274-328.

Wattenmaker, W. D., Dewey, G. I., Murphy, T. D., & Medin, D. L. (1986). Linear separability and concept learning: Context, relational properties, and concept naturalness. *Cognitive psychology*, *18*(2), 158-194.

Werker, J. F., Fennell, C. T., Corcoran, K. M., & Stager, C. L. (2002). Infants' ability to learn phonetically similar words: Effects of age and vocabulary size. *Infancy*, *3*(1), 1-30.

Willshaw, D. J., Buneman, O. P., & Longuet-Higgins, H. C. (1969). Non-holographic associative memory. *Nature*, *222*(5197), 960-962.

Wu, J.-X., Huang, P.-T., Li, C.-M., & Lin, C.-H. (2018). Bidirectional hetero-associative memory network with flexible sensors and cloud computing for blood leakage detection in intravenous and dialysis therapy. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*(4), 298-307.

Yamazaki, K., Vo-Ho, V.-K., Bulsara, D., & Le, N. (2022). Spiking neural networks and their applications: A review. *Brain Sciences*, *12*(7), 863.

Yáñez-Márquez, C., López-Yáñez, I., Aldape-Pérez, M., Camacho-Nieto, O., Argüelles-Cruz, A. J., & Villuendas-Rey, Y. (2018). Theoretical foundations for the alpha-beta associative

memories: 10 years of derived extensions, models, and applications. *Neural Processing Letters*, *48*, 811-847.

Yang, G. R., & Molano-Mazón, M. (2021). Towards the next generation of recurrent network models for cognitive neuroscience. *Current opinion in neurobiology*, *70*, 182-192.

Yang, G. R., & Wang, X.-J. (2020). Artificial neural networks for neuroscientists: a primer. *Neuron*, *107*(6), 1048-1070.

Yano, Y., & Osana, Y. (2009). Chaotic complex-valued bidirectional associative memory. 2009 International Joint Conference on Neural Networks,

Yoshida, K. A., Fennell, C. T., Swingley, D., & Werker, J. F. (2009). Fourteen-month-old infants learn similar-sounding words. *Developmental science*, *12*(3), 412-418.

Zhang, J., Zhu, S., Bao, G., Liu, X., & Wen, S. (2021). Analysis and design of multivalued high-capacity associative memories based on delayed recurrent neural networks. *IEEE Transactions on Cybernetics*, *52*(12), 12989-13000.

Zhang, X.-J., Moore, J. M., Yan, G., & Li, X. (2023). Universal structural patterns in sparse recurrent neural networks. *Communications Physics*, *6*(1), 243.

Zhou, Q., Wu, X., Zhang, S., Kang, B., Ge, Z., & Latecki, L. J. (2022). Contextual ensemble network for semantic segmentation. *Pattern Recognition*, *122*, 108290.

Zilly, J., Achille, A., Censi, A., & Frazzoli, E. (2021). On plasticity, invariance, and mutually frozen weights in sequential task learning. *Advances in neural information processing systems*, *34*, 12386-12399.