Multi-Modal Associative Memory: A framework for Artificial Intelligence

Rodrigo Simas, *Instituto Superior Técnico, Lisbon, Portugal,*

Abstract

Drawing from memory the face of a friend you have not seen in years is a difficult task. However, if you happen to cross paths, you would easily recognize each other. The biological memory is equipped with an impressive compression algorithm that can store the essential, and then infer the details to match the perception. The Willshaw Network (WN), is a model that aims to mimic these mechanisms of its biological counterpart. The usage of this model in practical applications is hindered by the so-called *Sparse Coding Problem*. To advance, prescriptions that transform raw data into sparse distributed representations are required. In this work, we use a recently proposed prescription [1] that maps visual patterns into binary feature maps. We analyze the behavior of the WN on real-world data and gain key insights into the strengths and weaknesses of this model. To further enhance the capabilities of the WN, we propose the Multi-Modal framework. In this new setting, the memory stores several modalities (e.g., visual, or textual) simultaneously. After training, the model can be used to infer missing modalities when just a subset is perceived, thus unlocking many practical applications. As a proof of concept, we perform experiments on the MNIST dataset. By storing both the images and labels as modalities, we were able to successfully perform retrieval, classification, and generation with a single model. Our results highlight the flexibility of the Multi-Modal framework to perform various classical Machine Learning tasks with a single network and provide a big hint on how the field of Associative Memories can advance in practical settings.

Index Terms

Associate Memory, Auto-Association, Willshaw Network, Generation,s Classification

I. Introduction

To create intelligent machines, understanding the human brain and its mechanisms is a fundamental step. For this reason, the focus of this work was the field of AMs: a family of Biologically-Inspired Artificial Intelligence models that imitate biological memories [2].

These models train by storing associations between pairs of patterns: Correlated features form synaptic connections between the memory's neurons. This way, a pattern is represented in the memory, by the activation of a population of neurons, and each individual neuron can participate in many populations. A trained memory can then be queried with a pattern: the neurons in the network will fire according to their learned connections, and the resulting population of active neurons will correspond to the memory's response to the query. This task is known as *Retrieval*, and the fact that both the query and the answer are patterns means that the memory is employing Content-Addressability.

In this work, we focus on the WN memory. This model is a single-layered Neural Network that uses a local Hebbian training rule [3] to store associations between pairs of binary vectors. Despite its simplicity, the model is extremely efficient in terms of its storage capacity, being able to store large numbers of patterns with a fixed amount of neurons, and a small computational effort.

Furthermore, we focus on the task of Auto-Association, i.e. we train the WN to associate patterns back to themselves (similarly to an AutoEncoder). An Auto-Associative memory has great practical uses, such as removing noise from corrupted patterns or reconstructing the whole pattern when only part of it is presented to the memory.

A. Problem

Our brains "(...) represent information using a method called SDRs" [4]. Neuroscience has shown that information in the brain is represented by the sparse activation of groups of neurons in the cortex [5].

In computers, an SDR is a large binary vector, where only a select number of bits are active. When representing complex real-world data, each active bit in the SDR must represent an important feature of the pattern. In a sense, SDRs are informative compressions, and AMs benefits immensely from this property due to their content-addressable nature.

The biggest open question in the field of AMs, which often prevents these models from being used in practice, is the so-called *Sparse Coding Problem*. In short, the memory's performance can only be ensured if the patterns that it stores are SDRs, and this type of representation rarely occur naturally.

To solve the *Sparse Coding Problem* and use the WN on practical settings, encoding prescriptions that transform real-world data into SDRs are required.

In [1] Sá Couto and Wichert propose one of the first sparse encoding prescriptions that addresses this problem. In their work, the "What-Where" (WW) Encoder is proposed: a biologically-inspired module that transforms visual patterns from the MNIST dataset of hand-written digits [6] into binary SDRs. In the same paper, preliminary research showed that the codes produced by the WW encoder could very efficiently be stored in auto-association on the WN. As a consequence, the opportunity to study the WN in a practical real-world setting emerged.

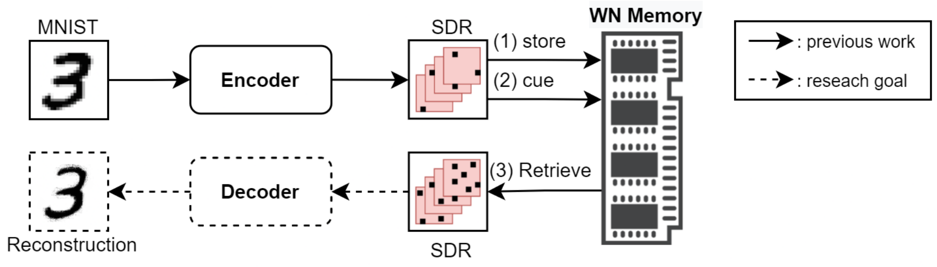


Fig. 1. **Initial Research Goal**. In [1] an Encoder that maps MNIST images into SDRs is proposed. These SDRs are stored in the Willshaw Network **(1)**, and retrieved **(2, and 3)**. To complete the pipeline, a Decoder module that transforms the outputs of the memory back into images is required.

B. Research Goals

Our first Research Goal was to perform a thorough analysis of the storage and retrieval of MNIST images in a Willshaw Network. This entailed building a decoder module that transformed SDRs back into images (Fig. 1). With an encoder and decoder built, we would be able to study the behavior of the WN on real data and perform experiments such as: (a) Visualise the quality of the memory's reconstructions as the memory is filled up; and (b) Visually examine how the memory responds to noisy cues. Additionally, a trained memory and the Decoding module could be used as a generative model: By querying the memory with novel inputs, one might obtain novel responses from the memory. Decoding said responses back into images might lead to original patterns. Essentially drawing from memory.

To build the decoder module, we tried using Feed-Forward Neural Networks (the MultiLayer Perceptron (MLP), and Convolutional neural networks (CNNs)), and a simpler nativist approach (the What-Where (WW) Decoder). The latter achieved the best results, and thus an extensive analysis of the WN model was made.

This analysis gave us key insights into the strengths, weaknesses, and possible applications of the WN in a practical scenario.

Motivated by the findings of the aforementioned analysis, the crux of this work emerges: The idea of the **Multi-Modal Willshaw Network (MMWN)** is proposed. In short, the memory stores several modalities simultaneously. After training, the model can be used to retrieve missing modalities when just a subset is shown, thus unlocking many practical applications. Where a modality refers to a type of information, e.g. visual or textual.

While simple, this idea has the potential to serve as a flexible framework to solve various Artificial Intelligence Tasks with Associative Memories. Demonstrating the feasibility of this idea became the main Research Goal of this work.

As a proof of concept for the idea of the MMWN, this model was evaluated on the MNIST dataset [6]. By storing both the images and labels as modalities, we were able to perform retrieval, classification, and generation successfully.

The relevance of the MMWN model can be summarized in the following arguments:

- The model has a simple Hebbian training rule that leans in a single pass over the training set and is highly efficient in terms of its storage capacity.
- The model represents information in a distributed manner, thus circumventing the "grandmother cell" problem that is often faced with Neural Networks.
- The model's architecture is not constrained to a particular domain, and a model can hold information from multiple domains (outside the scope of this work) simultaneously.
- The model solves several Machine Learning tasks (Retrieval, Classification, and Generation) in a single architecture, while others such as (CNNs, Variational Autoencoders (VAEs), or Generative Adversarial Networks (GANs)) specialize in a single task.
- The model unifies several ideas from biology: (a) It utilizes SDRs, the language of information in the neocortex; (b) The WN uses a biologically plausible learning rule, and mimics the biological Associative Memory; and (c) The new architecture exhibits the flexibility to perform several tasks that are deemed essential in Intelligent Systems.

With this work, we will demonstrate the potential of the Willshaw Network Model for practical applications. Furthermore, we propose a novel framework that allows further research to be done in the field of AMs.

II. Background

A. Associative Memory

Associative Memories (AMs) are a family of Artificial Neural Networks (ANNs) that store associations between pairs of vectors. If we consider these vectors to be abstractions of concepts or objects, learning and storing said associations closely resembles the way human memory operates. As Daniel Kahneham, Nobel prize winner puts it: "Psychologists think of ideas as nodes in a vast network, called associative memory, in which each idea is linked to many others." [2]. We can introspectively realize that our memory works this way when we try to recall some piece of information and notice that to reach the answer we must traverse a chain of associations. In this way, AMs are biologically inspired computational models in the sense that they try to incorporate these high-level functions of biological memories. Additionally, these models implement neuro-physiological mechanisms of the brain such as the rule for synaptic plasticity proposed by Donald Hebb [3].

It is important to underline that AMs are quite different from traditional artificial memories, like those found in computer disks. In a computer, a memory is a list of addressed information: we provide an address and obtain its content back, the memory employs *location addressing*. AMs do not utilise addresses explicitly, we provide a *question* content vector and obtain an *answer* content vector back. As explained by Palm [7], AMs are *mapping memories* and they support *content addressability*.

Formally, an AM stores a set S of M pairs (x, y), where x and y are the *question* vector and the *answer* vector respectively:

$$S = \{(x^{\mu}, y^{\mu}) \mid \mu = 1, \ldots, M\} \tag{1}$$

The AM establishes a mapping $(\mathrm{x}^{\mu} \rightarrow \mathrm{y}^{\mu})$ which is denoted *hetero-association*. In the special case where $x = y$ the memory is said to perform *auto-association*. Mapping a vector to itself in auto-association has great practical uses. Namely, removing noise from corrupted patterns or reconstructing the whole pattern when only part of it is presented to the memory. The pairs of patterns are not stored explicitly in the memory, it is the correlations and anti-correlations between the pairs of patterns that are learned. This key feature of AMs acts as a natural regularization mechanism and allows a trained network to generalize for novel patterns that are not stored in memory. The combination of hetero and auto association capabilities of an AM allows it to naturally implement functions of biological memories. The abilities of AMs have been studied for over fifty years now by many authors such as Palm [8] and Kohonen [9]. In [10], Wichert summarizes the core abilities of AMs as follows:

- The ability to correct false information.
- The ability to complete information in case only part of it is known.
- The ability to interpolate information: When novel information is shown, the most similar stored sub-symbol is determined.

The two main steps involved in operating an AM are *learning* and *retrieval*. Learning is the step in which the set of associations is stored and a mapping established. In the retrieval step, the mapping is used to perform association. Several types of AMs implement these steps differently. Here we skip the details and focus on the overall idea of each step.

1) Learning: An AM is typically represented as a *synaptic weight matrix* W of real numbers. In a network with n neurons each having m connections, W will represent the *weights* of the $m \times n$ connections between all neurons. Initially the memory will be empty ($W_{ij} = 0 : i = 1, \ldots, m; j = 1, \ldots, n$). The learning process consists in presenting M pairs (x^{μ}, y^{μ}) to the network, which can result in local updates to the synaptic weights of each individual neuron:

$$W_{ij} = \sum_{\mu=1}^{M} R\left(x_i^{\mu}, y_j^{\mu}\right) \tag{2}$$

Where R is the *synaptic weight update expression*, which takes as input the pre-synaptic and post-synaptic signals x_i and y_i, respectively, and outputs the updated weight value W_{ij}. An example of a *synaptic weight update expression* is the Hebbian rule [3] for binary vectors:

$$R\left(x_i, y_j\right) = x_i . y_j \tag{3}$$

The two major families of learning rules are *additive* and *binary*. In additive learning W is updated successively with Eq. (2). In binary learning the weight matrix is clipped so that a simpler binary matrix is obtained at the cost of some information loss:

$$W_{ij} = \sum_{\mu=1}^{m} H\left(R\left(x_i^{\mu}, y_j^{\mu}\right)\right) \tag{4}$$

where H is the *Heaviside* function defined as:

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{5}$$

In the AM that we will study, the Willshaw Network (WN), both the pairs (x, y) and the W matrix are binary. This means that a binary learning rule is employed.

2) Retrieval: Associative retrieval is the process of showing a *cue vector* to the already-trained memory, letting the network compute its final state, and obtaining the *retrieved vector*. Fig. 2 illustrates this process.

When the AM receives as input a *cue vector* $\tilde{\mathbf{x}}$, each neuron of the network will compute its dendritic potential s_j.

$$s_j = \sum_{i=1}^{m} W_{ij} \tilde{x}_i \tag{6}$$

Notice that this step can be parallelized since each neuron works independently of the others.

Next, each neuron computes its output $\hat{y}_j$ by applying a *transfer function* f, with threshold θ_j, to its previously computed dendritic potential.

$$\hat{y}_j = f\left(s_j, \theta_j\right) \tag{7}$$

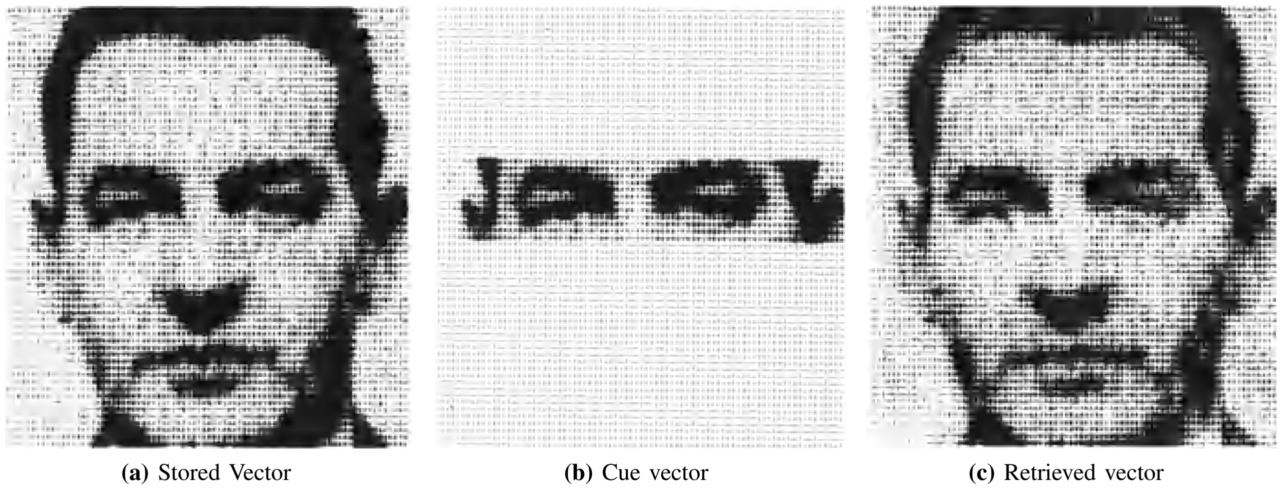


Fig. 2. **Example of associative retrieval**. A memory that stores 160 different pictures of faces, when given a small snippet containing just the eyes, is able to retrieve the original image. **(a)** One example out of the 160 images stored in the memory. **(b)** The cue vector that is used as input to the retrieval process. **(c)** The retrieved vector that is obtained after the memory computes its final state. Adapted from [9], original study in [11].

The *transfer function* f usually takes the form of a *step function*, like the *Heaviside* function described in Eq. (5). The choice of the *threshold* parameter θ, which controls the sensitivity of activation, is of extreme importance: an appropriate rule can greatly enhance the memory's performance, allowing for the storage of more vectors with no penalty on the retrieval quality. If θ is set too high, very few neurons will activate, even with strong dendritic potential. On the other hand, if θ is set too low, many neurons will activate in a "false-positive" manner. A well-suited value for θ will ensure that the correct amount of correlations between neurons and the *cue vector* are detected, resulting in an error-free *retrieval*.

Some rules define a global *threshold* $\theta_j = \theta_{global} \mid j = 1, \ldots, n$ shared by all neurons, others rely on each neuron to store its activation threshold locally.

After all the neurons have computed their state y_j, the state of the whole network represents the retrieved vector y. When the retrieval process ends here, it is called *one-step retrieval* which occurs on feed-forward AMs. However, certain AMs, like the Hopfield Network [12], employ *recurrent* architectures. In such cases, the state y computed in the first step of *retrieval* is fed back into the network as input, and the retrieval process is repeated until convergence. This method is known as *iterative retrieval*.

3) Performance Assessment: The performance metric of associative memories is its *storage capacity*. After all, a memory system is as good as the number of different memories it can hold. Any associative memory can hold a small number of vectors and retrieve them perfectly, but as more vectors are stored, the memory starts to fill up and retrieval becomes imperfect (see Fig. 3 for an illustration). If too many vectors are added to the memory it can become saturated and rendered useless, not even able to retrieve the first learned vectors.

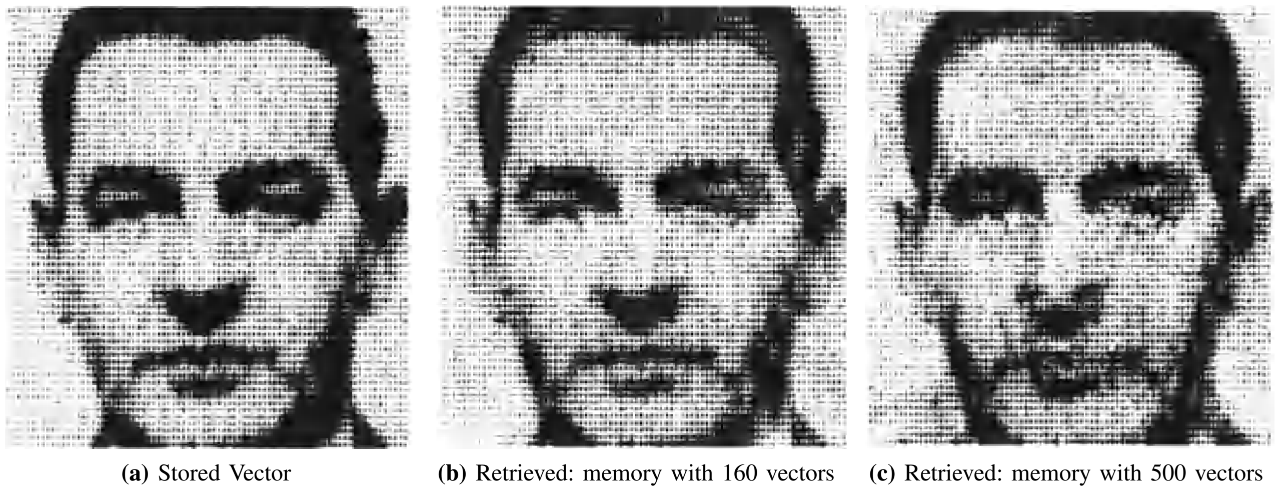
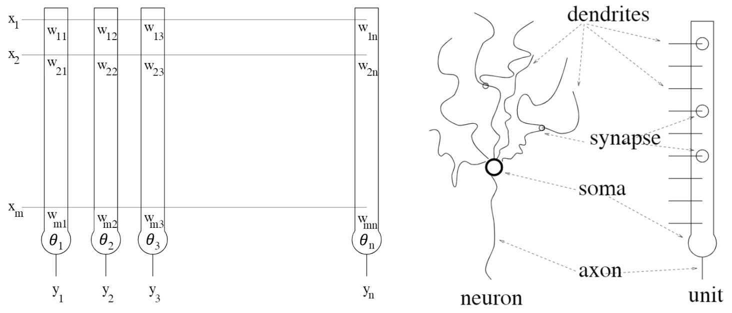

(a) Stored Vector
(b) Retrieved: memory with 160 vectors
(c) Retrieved: memory with 500 vectors

Fig. 3. **Effect of the number of stored vectors on retrieval quality**. In this experiment, similarly to Fig. 2, a small part of the image is used as the cue vector for retrieval. **(a)** The original vector that is stored in memory. **(b)** The retrieved vector obtained when the memory is filled with 160 vectors. We can notice some small imperfections in the retrieved image. **(c)** The retrieved vector obtained when the memory is filled with 500 vectors. The high number of stored vectors is causing a large retrieval error, the retrieved image has noticeable artifacts. Adapted from [9], original study in [11].


x_1
x_2
x_m
w_{11} w_{12} w_{13} w_{1n}
w_{21} w_{22} w_{23} w_{2n}
w_{m1} w_{m2} w_{m3} w_{mn}
θ_1 θ_2 θ_3 θ_n
y_1 y_2 y_3 y_n
dendrites
synapse
soma
axon
neuron
unit

(a) Willshaw Network schematic. **(b)** Biological vs Artificial neuron.

Fig. 4. **Neurons of the Willshaw Network**. **(a)** A schematic representation of a *Willshaw Nework*. Composed of a set of n vertical units which represent a simple model of a real biological neuron. Each neuron has m weights, which correspond to the synapses and dendrites in the real neuron. In this Figure they are described by $w_{ij} \in \{0,1\}$ where $1 \leq i \leq m$ and $1 \leq j \leq n$. θ_j is the threshold of each neuron j. **(b)** A side-by-side comparison of a biological neuron (left) and a Willshaw Netork's unit (right). Adapted from [10].

Formally, the storage capacity L, admitting an *error factor* ϵ of an AM with n neurons, is the total number of pairs (x,y) that can be stored in memory without experiencing a retrieval error greater than ϵ. Typically described as a function of the size n (number of neurons) of the network.

For example, the Linear Associator (studied independently by [13], [14], [15], or [16]), a very simple AM, had a storage capacity of $L = n$ when storing orthogonal vectors. Another example is the original version of the Hopfield Network, which had a capacity of $L = 0.14n$ for uncorrelated vectors. Notice that these theoretical values for the storage capacities of AMs are often studied under ideal scenarios, such as storing uncorrelated or orthogonal vectors. As it turns out, vectors with certain properties are very well suited for storage in AMs, resulting in great storage capacities. However, real-world data typically lacks those properties, which is the main cause for the shortcomings of AMs in practical applications. Finding mechanisms to transform real-world data into "AM-suitable" vectors is the key to unlocking the full potential of AMs.

B. Willshaw network

The Willshaw Network (WN) [17] is a shallow, feed-forward ANN that performs *auto* and *hetero-association* of binary vectors.

Originally named *Lernmatrix* by its creator Karl Steinbuch in the 1950s [18], the biologically-inspired model was one of the pioneer implementations of ANNs. The Lernmatrix became more popular in the following decade when it was studied through a mathematical/biological lens by D. Willshaw [17]. Thus becoming known as the *Willshaw Network*. Despite its simplicity, both in its binary nature and in its learning/retrieval rules, the WN is a great artificial memory, capable of storing large numbers of vectors.

1) Architecture: The WN is composed of n units, also referred to as *neurons*. Each neuron is connected to the input layer through m binary connections. Neurons are not interconnected. A schematic view of the network can be seen in Fig. 4(a).

The network is represented as a binary matrix W:

$$W_{ij} \in \{0,1\} : i = 1, \ldots, m; j = 1, \ldots, n \tag{8}$$

The dimensions m and n are given by the fixed sizes of the *question* and *answer* vectors, respectively. Notice that, in an auto-association task, W will be a square matrix of size $(n \times n)$. This matrix defines the absence/presence of connections between neurons. A connection between two neurons i and j is formed during training when a local Hebbian rule detects a correlation between the positions i of a *question* vector, and the position j of an *answer* vector.

Additionally, each neuron has a local threshold T_n, used in the last step of retrieval as the parameter of the *transfer function*.

2) Biological plausibility: The human brain is the most intelligent system that we know of. Understanding the brain and implementing its mechanisms is the ultimate goal of Biologically inspired models. AMs are examples of such models as they try to imitate the associative capabilities of the brain.

The *Lernmatrix/Willshaw Network*'s development has always been guided by biology and neuroscience. The original purpose of the model was to study psychological conditioning [18], [19]. The model was later analyzed by Palm [20] on its ability to implement assemblies of cells under the Hebbian framework [3]. Furthermore, the network's units are artificial neurons [21] modeled after their biological counterparts (see Fig. 4(b)). Finally, the neural energy efficiency of the model has been analyzed [22], [23].

All in all, the WN, while far from a replica of the brain, is heavily biologically inspired. Consequently, its application in practical scenarios is significant, since it might give us interesting insights into our still limited understanding of the brain.

3) Learning and Retrieval rules: The WN is completely binary: its inputs x, outputs y, and weight matrix W are binary. Consequently, its learning and retrieval rules are simple.

For the next equations, let us use the following convention:

- The n neurons of the network are referenced with the index j.
- The m connections of each neuron are referenced with the index i.
- The weight matrix W has m rows and n columns (see Fig. 4(a)), and it is indexed as W_{ij}.

The learning rule is directly derived from Eq. (4). It uses the Hebbian rule (Eq. (3)) as its *synaptic weight update expression*. Given a set of M pairs (x, y), the weight matrix W is computed with the following binary learning rule:

$$W_{ij} = \sum_{\mu=1}^{m} H\left(x_i^\mu y_j^\mu\right) = \min\left(1, \sum_{\mu=1}^{M} x_i^\mu y_j^\mu\right) \tag{9}$$

notice that the Heaviside function $H(x)$ (Eq. (5)) is equivalent to $\min(1, x)$ due to the binary nature of the model.

The retrieval of a cue vector $\tilde{x}$ is performed in two steps. First the dendritic potential s is computed in each neuron:

$$s_j = \sum_{i=1}^{m} W_{ij}\tilde{x}_i \tag{10}$$

Secondly, the retrieved vector $\hat{y}$ is computed for all neurons:

$$\theta_j = \max_{1 \le i \le n} s_i \tag{11a}$$

$$\hat{y}_j = H\left(s_j - \theta_j\right) \tag{11b}$$

Eq. (11a) is known as *soft thresholding*. It defines a global strategy for the transfer function's activation threshold θ. This strategy was used by Palm et al., Schwenker, Sommer, and Strey in [24], [25]. Soft thresholding is the best performing strategy for the WN. Alternatively, one could use a simpler strategy, such as *hard thresholding*, where θ is set globally to the number of ones in the cue vector: $(\theta_j = \sum_{i=1}^{m} \tilde{x}_i \mid j = 1, \ldots, n)$. In Eq. (11b), H refers to the *Heaviside* step function described in Eq. (5).

Eqs. (9) to (11) fully specify the computations required to operate a WN.

C. Multi-Modal Willshaw Network

The WN makes no assumption about the data that it stores. Each bit of a pattern that is stored in a WN is interpreted as an informative feature. The position of a particular feature within the pattern, and its meaning/interpretation are irrelevant to the memory. As a result, we can fill the memory with heterogeneous patterns; i.e., patterns containing multiple modalities/types of information, as seen in Fig. 5. In such cases, we are building a Multi-Modal Associative Memory (MMWN).

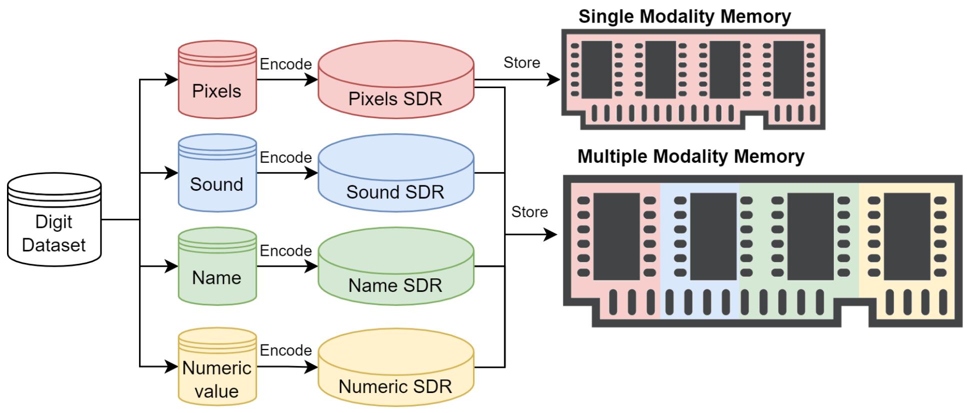


Fig. 5. Multiple vs Single modality memory. In this example, we are storing digits in both memories. If we consider a single type of information about the digit, then we require a Single Modality AM(top). If we consider multiple types of information about each pattern, we use a Multiple Modality AM(bottom).

One of the main strengths of the WN is the ability to complete missing information. A MMWN benefits immensely from this property: During training, the memory will learn how all the different features, from all modalities, are correlated. On retrieval, all the different modalities contribute to the memory's response. If the information from one of the modalities is missing in the retrieval cue, the memory can complete it using the remaining modalities.

D. Sparse Encoding Prescriptions

Strategies that transform natural data into SDRs are required to solve the sparse coding problem. Here, we refer to such strategies as *encoding functions* or *prescriptions*. The desired properties of an *encoding function* are [4]:

- Semantically similar data should result in SDRs with overlapping active bits.
- The same input should always produce the same SDR as output.
- The output should have the same dimensionality (total number of bits) for all inputs.
- The output should have similar sparsity for all inputs and have enough one-bits to handle noise and subsampling.

An important aspect of Artificial Intelligence (AI) models is their amount of predefined *structure*. AI models can be broadly divided into two categories: those that use "a priori" knowledge in their structure (*Nativism*), and those that don't (*Connectionism*). The first is a traditional approach [26], where the knowledge about the domain of the problem is used to design comprehensible solutions. The second is focused on building general algorithms (usually Neural Networks) that can learn with minimal assumptions/biases about the problem. Due to the huge diversity of available AI techniques, there isn't a binary way to classify models in terms of their structure, instead, there is a wide spectrum. In the *Nativism* end of the spectrum, we have techniques such as first-order logic models [27] where the model's structure is strictly predefined. On the other end, we have *Connectionism* techniques, such as the multi-layered perceptron (MLP) [28], where the structure is not imposed beforehand, but learned through the weights of the model. In between these two extremes, there is a large diversity of models that have both Nativism and Connectionism aspects. From Deep Blue [29], the chess machine that beat the world champion with an efficient tree search, to a deep reinforcement learning model that learns to play different video games using only the raw pixels as input [30].

E. What-Where Encoder

"Vision allows humans to perceive and understand the world surrounding them, while computer vision aims to duplicate the effect of human vision by electronically perceiving and understanding an image." [31] The field of Computer Vision has been highly connected to biology [32] since the 1980s. The field's Holy Grail is to build a model that performs as well as the human brain. For this reason, many *Nativism* models, that implement our biological knowledge of vision, arise in image processing tasks.

In [1], Sa-Couto & Witchert propose an *encoding function* that maps visual patterns into informative binary vectors. The *Nativist* model is biologically inspired, using similar ideas to those of the *Neocognitron* [33]. The *Neocognitron* is a direct implementation of Hubel & Weisel's Nobel-Prize-winning theory of the mammalian visual cortex [34], and it is the precursor of the widely popular Convolutional Neural Networks (CNNs) [35].

The goal of the authors was to implement a sparse encoding function for visual patterns that could be used to pre-process data in classification tasks. The resulting representations were SDRs, so the authors experimented on the Willshaw Network (WN): This Associative memory will have a better storage capacity if the representations it receives are informative. Knowing this, we can evaluate the quality of the encoding prescription as a whole by monitoring the retrieval score of a WN that stores the representations.

Using an AM to store such a large collection of natural data is traditionally a very hard task for these models, and as expected, retrieval was imperfect. For this reason, the authors proposed a novel error measure that utilized the labels of the patterns. The results show that the codes produced by this strategy are effectively compressing the information in the patterns, so much so that a very simple 1 nearest-neighbor classifier can perform classification when trained on the outputs of a WN that stores the compressions in auto-association.

The strategy has two main steps:

The retinotopic Step: A convolution-like layer [36] of units that act as receptive fields are locally connected to the input layer. These receptive fields detect and extract the most relevant visual features of the patterns. These features are determined beforehand with the unsupervised K-means algorithm [37], [38]. This layer performs information compression by establishing a many-to-one relationship between groups of pixels and receptive units. In this way, it transforms the dense representation presented in the input layer into a sparse representation.

The Object-Dependent Step: The fixed coordinate system of the representations is turned into an object-dependent, radius one polar coordinate system. This object-oriented transformation is known to occur in mammalian brains [39]. For this step, only the radius and center of each object need to be measured, no learning is required. This operation provides invariance to size and position, and in turn, also makes the resulting representations well-distributed since the features detected in the previous step will be mapped quasi-uniformly into all the dimensions of the object space.

The encoding process can be controlled with the following set of parameters: K specifies the number of visual features that will be extracted with the K-means algorithm, the Field size (F_s) specifies the "radius" of the visual features (which are square windows of side $2F_s + 1$), Q represents the size of the new coordinate system plane and, finally, the threshold T_w controls the level of similarity needed to signal the presence of a feature in an image. The former can be used to tune the sparsity of the codes.

The encoder receives the raw MNIST dataset with dimensions (N, 28, 28), where N is the number of patterns. The resulting sparse codes are binary vectors with dimensions (N, $Q \times Q$, K), i.e., each one of the N patterns is represented as a set of K features maps of size $Q \times Q$.

F. The What-Where Decoder

The What-Where (WW) Decoder implements the inverse function of the WW encoder. The decoder transforms the sparse code set of shape $(N, Q \times Q, K)$ back into the original MNIST shape $(N, 28, 28)$, where N is the number of patterns in the dataset, Q is the size of the polar coordinate system, K is the number of visual features learned with the K-means algorithm, and 28 is the original size of the MNIST images. The decoding strategy has three steps:

The Object dependent Inverse step: The object-dependent, radius one polar, $Q \times Q$ coordinate system is converted back into the Cartesian, 28×28 coordinate system. To do so, the radius and center of the object (which were measured in the encoding process) are used. If the radius and center of the patterns were not saved, the default values of $c = (0, 0)$ and $r = 1$ are used. This step changes the dimensions of the codes from $(N, Q \times Q, K)$ into $(N, 28 \times 28, K)$, where K is the number of visual features.

The Retinotopic Inverse Step: The outputs of the last step are a set of K binary feature maps that signal the presence of visual features. In this step, we transform these bits into the actual visual features by performing a "transposed convolution" where the convolution kernels are the ones learned with the K-means algorithm before the encoding process. Since the WW encoder uses a stride of 1, same-padding, and square kernels with odd sizes, this step is straightforward: an active bit on plane k and position (i, j) will result in the visual feature k being drawn with its center on position (i, j) of the MNIST images space. The output of this layer has the same dimensions as the original MNIST images.

Normalization Step: The former step can produce images where the value of a particular pixel is greater than 1 (recall that we represent the MNIST images as arrays of pixels valued between 0 and 1). This occurs when two or more features are detected in close proximity, resulting in some pixels being drawn by multiple sources. To circumvent this issue, we count how many visual features participate in the drawing of each pixel of the image, and then divide the value of the pixel by the number of features that contribute to it.

G. The biology of Encoding Functions

One might ask: "If Associative memories are biologically plausible, why is their performance on raw data poor? The brain handles raw data just fine." This is a valid comment, and in this section, I will give a hint on the motivation behind encoding prescriptions.

Representing human knowledge on a computer has proven to be a difficult task. The problem stems from the fact that our knowledge of the world is not well-organized. Every rule has exceptions and every fact links to numerous others. This level of complexity is not well-suited for traditional computer data structures. In contrast, our brains seem to deal with this complicated knowledge with ease. Our brains do not use dense representations like those used in computers. Instead, they represent knowledge with the sparse activation of its billions of neurons. The brain uses SDRs to encode knowledge [4]. This might sound contradicting. After all, if real-world data, such as images, is dense, how can the brain use SDRs? Indeed, sensory data is not naturally found in the form of SDRs. But we know that our brain has lower functional regions that process the inputs from the senses and generate sparse and distributed activation of neurons in the neocortex [5]. For instance, the visual area of the brain is divided into several hierarchical regions: V1, V2, V4, and IT. The V1 area of the brain is responsible for detecting low-level visual features such as edges, and basic color [40]. The detection of such features results in the activation of a collection of neurons that forms a signal. This signal is passed onto the V2 area which will apply a similar process. After passing through all the regions, the final result will be an SDR that represents what we are sensing [41]. Another example is auditory perception. The sounds that we hear are captured in the cochlea. The cochlea is an organ that has several receptors for different frequencies. A particular sound will result in the sparse activation of the receptors which will result in an SDR.

All-in-all there is a biological analogy between SDR encoders used for Associative memories and biological organs that transform sensory data into sparse neuron activity in biological memories.

III. Results

In this section, we will demonstrate that the MMWN can work in practice, on real data. To do so, we will use the well-known MNIST dataset of hand-written digits [6] to train a MMWN with two modalities and use it for practical applications.

Each pattern in the MNIST dataset has two attributes: the label represented by an integer; and a picture of the hand-written digit represented as an array of pixels.

Our previous work [1] has shown that a normal WN can efficiently store and retrieve the images of the MNIST dataset. In this work, we will use a MMWN that stores both the images and the labels of the MNIST dataset. If the MMWN manages to store both modalities efficiently, we can leverage the information-completion properties of the memory for practical applications such as the generation of new patterns and classification.

A. *Description modality*

To use the label of the digit (which is an integer between 0 and 9) as a modality, we must first transform it into a binary code. Furthermore, this binary code should be suitable for a WN; i.e., be an SDR. Next, we propose a stochastic encoding strategy that transforms integers into SDRs that we refer to as **descriptions**.

1) Noisy X-Hot Encoder: Our proposed encoding strategy is the Noisy-X-Hot (NXH) encoding, which is a stochastic version of the well-known X-Hot encoding. The main idea is that X bits in the code will randomly activate with high probability, while the remaining bits will be active with low probability. The result is an X-Hot encoding with some added noise/randomness, hence the name.

Formally, for a label l in the range $\{0, \dots, L-1\}$, its Noisy-X-hot encoding $e(l)$, with X bits per class, and probabilities P_{class}, and P_{rest}, is a binary array of size $L \times X$ given by:

$$e_i(l) = \begin{cases} 1 & \text{with } P_{class} \text{ probability, if } i \in \{z \in \mathbb{Z} \mid lX \leq z < (l+1)X\} \\ 1 & \text{with } P_{rest} \text{ probability, if } i \notin \{z \in \mathbb{Z} \mid lX \leq z < (l+1)X\} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

The resulting code can be thought of as a collection of L populations of neurons of size X, one for each of the L integer values we are encoding. For a given label l, its NXH code will have most of its activity on the l^{th} interval. Furthermore, two distinct patterns with identical labels ($l_1 = l_2$) will likely have different NXH encodings ($e(l_1) \neq e(l_2)$) due to the stochastic nature of the NXH. In this way, the NXH code is not only an encoding of the labels but also a **description** which allows us to differentiate between different patterns with the same label.

2) Noisy X-Hot Decoder: For practical purposes, we must have a mechanism that maps the NXH codes back into the integer value of the label. We can achieve this, by looking at the total activity of all the L intervals of the NXH code and picking the interval with the most activity.
Formally, a NXH code e, can be mapped back into its corresponding label l with:

$$l = \arg\max_i \left(\sum_{n=iX}^{(i+1)X} e_n \right) \tag{13}$$

B. *Methodology*

As we have seen in Section II-A, there are two key steps when operating an AM: the learning step and the retrieval step. The same is true for a MMWN.

The learning step consist in: (1) encoding the labels and pixels into the descriptions and codes, respectively; (2) concatenating the description and code into what we refer to as a desCode (DC) (short for description and code); and (3) storing the DC in auto-association, as depicted in Fig. 6

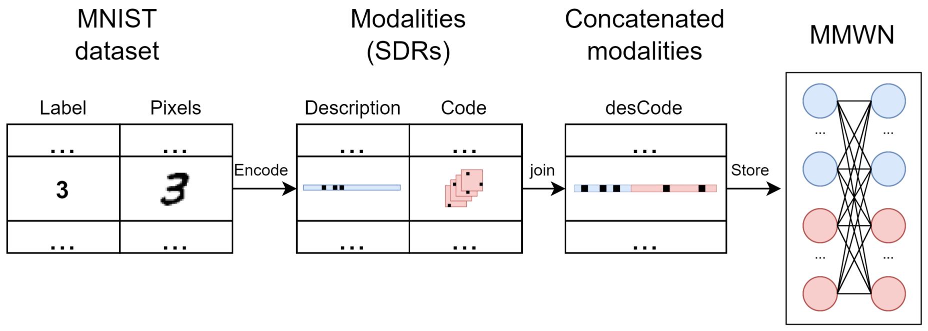


Fig. 6. Training Step in a MMWN.

The Retrieval Step simply consists in showing a cue DC to the trained memory and obtaining a retrieved DC back. One can, however, manipulate the cue in different ways to perform different tasks, as we will see in the upcoming sections.

C. *Classification*

One of the WN's greatest strengths is to complete missing information from the cues. One can take a set of DCs $\mathbf{X}$, set all the bits in the description modality to zero to create noisy cues $\tilde{\mathbf{X}}$, and obtain the memory's response $\mathbf{Y}$. The memory will complete the missing information, which in this case is the description modality. In this setting, the memory works as a classifier.

To test the MMWN's performance when it comes to classification we defined three tasks of increasing complexity: Auto-association, classification of stored patterns, and classification of unseen patterns (see Fig. 7). Results are reported in Fig. 8.

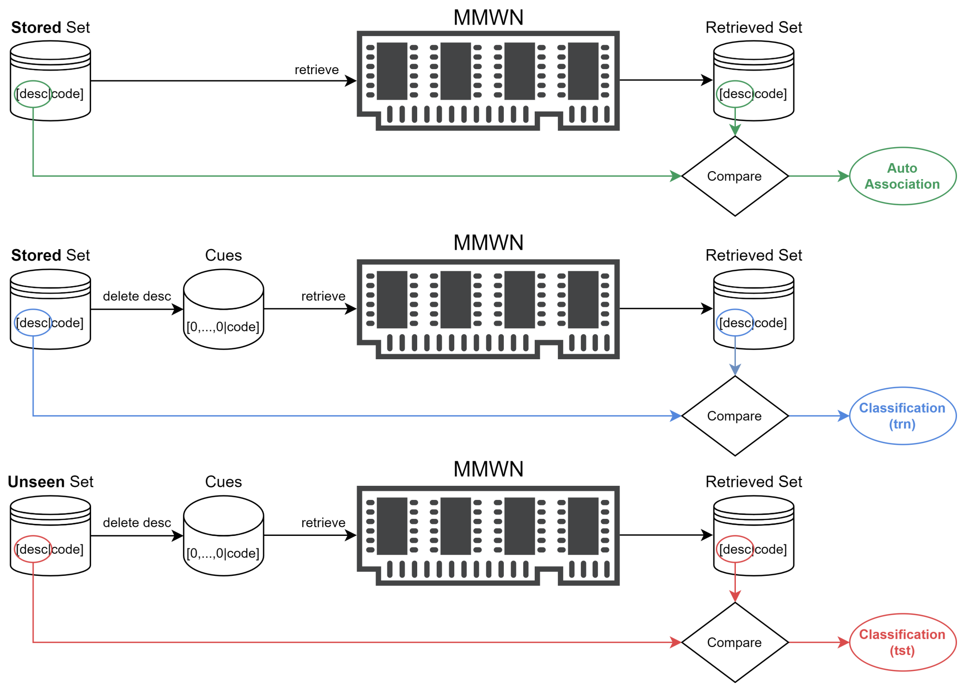


Fig. 7. **MMWN Classification methodology**. To evaluate the MMWN's ability to perform classification we define three distinct tasks of increasing difficulty: **(Top):** Auto-Association, **(Middle):** Classification of stored patterns, and **(Bottom):** Classification of unseen patterns. In all cases, we compare the original description of the pattern (before memory) with the description in the retrieved vector (after memory). For all three tasks, the accuracy score is determined by comparing the integer value of the label before and after the memory which is achieved by decoding the description with Eq. (13).

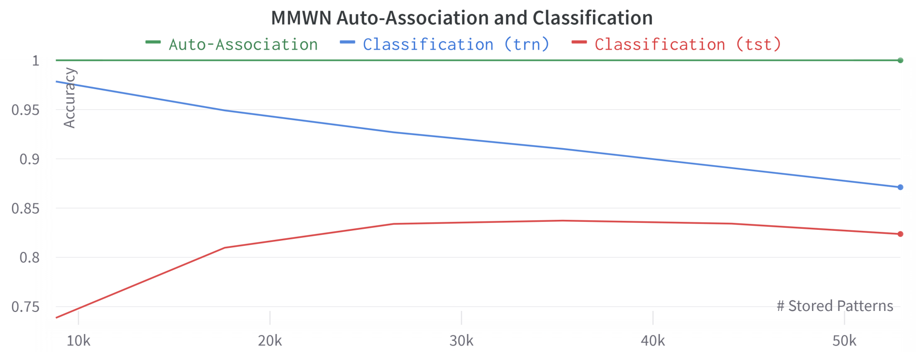


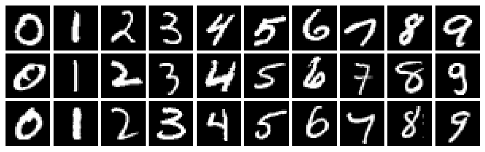
Fig. 8. **Auto-Association and Classification Results with the MMWN**. Here we follow the methodology of Fig. 7 to report the scores of the three tasks as the memory is filled with patterns. For the WW encoder, we used the parameters of the best performer in [1]. For the NXH encoder we used: $X = 500$, $P_{class} = 0.5$, and $P_{rest} = 0.0$. **(Green):** Auto-Association score. Here we are simply measuring the memory's ability to auto-associate the description of DCs. The accuracy is perfect even when the memory is full, indicating that the descriptions are tolerant to the noise that the memory introduces. **(Blue):** Classification of stored patterns. This score is high but decreases monotonically as the memory is filled up, which is expected since the memory gradually loses the ability to perfectly retrieve the patterns that it stores. **(Red):** The classification accuracy for unseen patterns improves as the memory stores more information because the memory naturally becomes better at generalizing as it stores more information (peeking at 84.04%). However, once the memory becomes too full, this score gets worse.

D. Generation

Inversely to classification, generation with a MMWN is performed by creating a set of cues $\tilde{\mathbf{X}}$ where the description modality remains intact but the visual modality is set to zero. The memory's response $\mathbf{Y}$ will complete the missing information in the visual modality of the DC, essentially **generating** a pattern from a description.

Providing a trained MMWN with a DC where the visual modality has been completely set to zero yields the results in Fig. 9(b). The description modality alone is not able to generate unique patterns that resemble examples from the original dataset, such as those in Fig. 9(a). Instead, a "blob" (Fig. 9(b)), with no detail, which looks as prototype of each class is obtained.

The amount of bits in a retrieved vector increases as we delete information from the retrieval cue (due to Eqs. (10), (11a) and (11b)). The "blobs" of Fig. 9(b) are obtained from retrieval cues that have no active bits in the visual modality, i.e. cues


(a) MNIST examples

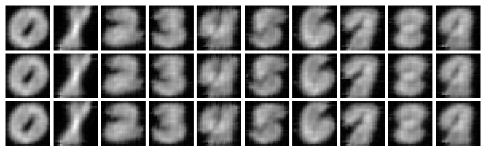

(b) Generated examples

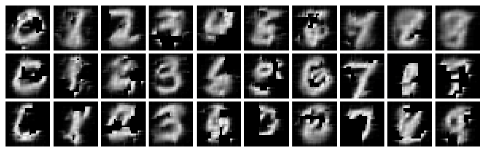
Fig. 9. **MNIST examples (a) and Drawings from memory (blobs) (b)**. The blogs are obtained when we provide a MMWN with cues that contain the description, but are missing the visual modality.

where a lot of information has been deleted. Consequently, the memory's response is overloaded with information, and the reconstructions have too many pixels.

To move away from "blobs" and create more realistic generations, we cannot provide zero information in the visual modality. Instead, we must "seed" the generation process by adding some visual information to the retrieval cue. **We hypothesize that:** a small amount of visual information together with the description of the pattern will allow the memory to leverage its information-completion capabilities, and generate more realistic patterns.

Next, we propose different ways to "seed" the generation process and report the resulting generations.

1) Naive Sampling-Based Generation: To test our **hypothesis**, we generate patterns using a naive sampling process: First, we compute the probability distribution of the visual modality of each class by counting the relative amount of ones for each position in the visual code. Then, we create an "artificial" code by independently sampling from the probability distribution. Lastly, we present the sample and a description from the class to the Multi-Modal Memory and decode its response. Fig. 10 compares the results of this sampling-based generation with a Single-Modality WN (Fig. 10(a)), and a MMWN (Fig. 10(b))


(a) Baseline Generation with Single Modality WN

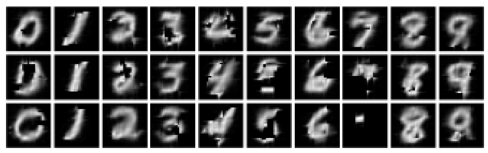

(b) Sampling-based Generation with a MMWN

Fig. 10. **Sampling-based Generation: Single vs Multiple Modality Memory**. Here we are generating images by sampling from the class distribution, retrieving the sample in a full memory, and reconstructing the memory's output with the What-Where decoder. **(a):** The memory is a Single-Modality Willshaw Network. **(b):** A Multi-Modal Willshaw Network (MMWN) that stores descriptions and visual codes is used. Please note that in both (a) and (b) we delete bits from the sample ($P_{del} = 0.5$). Using the new modality clearly improves the quality of the generations. However, the memory will, occasionally, corrupt the sample.

The addition of the description modality improved the generation quality. However, the memory will occasionally corrupt the sample by deleting most of the information in the pattern (for instance, the bottom-most 7 of Fig. 10(b)), or by adding too much information (bottom-most 9 of Fig. 10(b)). Next, we propose a simple modification to the sampling-based algorithm that circumvents this issue.

2) Trial-and-error Sampling-Based Generation: When the WN retrieves a cue, the number of bits in the retrieved vector is indicative of its quality. If we analyze the number of bits across many retrievals (Fig. 11) we can see that the number of bits in the retrieved vectors follows a normal-like distribution, where the tails of this distribution correspond to the cases where the memory corrupted the patterns.

We can improve the generation quality by forcing the generations to have a "normal" amount of bits via trial-and-error: If the generation created a pattern with a number of bits that falls into the tails of the distribution, we discard the generation and try again. If the generation creates a pattern with a number of bits that falls into the acceptance interval we finish the process. The effect of this simple modification on the generation quality can be seen in Fig. 12.

3) Iterative Generation approach: While the generation results from the previous strategies are promising, they rely on the computation of the probability distribution of each class to "seed" the generation process. Here we propose a different generation strategy that does not have this constraint.

The main idea is to seed the generation process using a sample from the "blobs" (Fig. 9(b)) instead of using a sample from the probability distribution of the class. Recall that the blobs are generations that are obtained when we simply provide the memory with a description and nothing else. By using the blob, the generation process is much more elegant, as it simply requires a description.

The blobs are visual codes that contain a lot of active bits, and they appear to be a sort of average of many examples from the class. If we delete most of the bits from a blob, we will be left with a small set of visual features that can be used as a retrieval cue for generation. Let us denote this process of sampling from the memory's output as **sparsification**.

Our final generation strategy is an iterative process that combines the idea of sparsification, with the idea of the trial-and-error generation (Section III-D2).

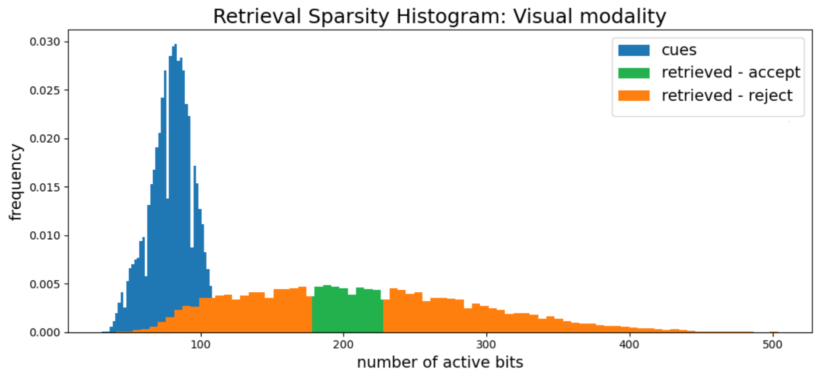
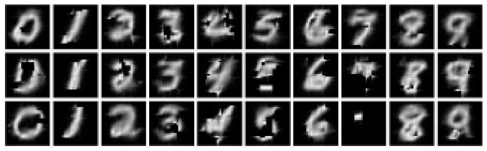
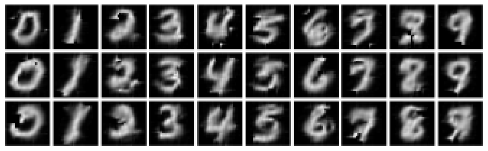


Fig. 11. **Retrieval: Number of bits histogram.** Here we used a trained memory to retrieve 30000 noisy cues. The number of bits in the cues, and retrieved vectors is measured and plotted. We can see that both the cues (blue) and retrieved vectors (orange/green) appear to follow a normal distribution. The number of bits in the retrieved vector can fall within a large range, depending on the quality of the cue. In the trial-and-error generation, we specify an acceptance interval (green): if the number of bits in the memory's output falls within the acceptance interval we keep the generation; if the number of bits ends up in the tails of the distribution (orange), we discard the generation and try again with a new cue.



(a) Sampling-based Generation with a MMWN



(b) Trial and error sampling-based Generation with a MMWN

Fig. 12. **Sampling-based Generation with a MMWN: Simple vs Trial-and-error approach**. Here we are generating images with the MMWN. First, we create a descriptor of the class and create an artificial code by sampling from the class distribution. Then we provide the artificial desCode to a full MMWN and reconstruct the memory's output with the What-Where decoder. **(a):** Simple generation, for each generation in this plot we used one sample and performed retrieval once. **(b):** Trial and Error Generation. Here we monitor the sparsity of the generations: if the number of bits of the retrieved vector is not within a predefined interval, we discard the generation and try again with a new sample. In this experiment, the generation process took an average of 4.63 attempts per generation.

The general idea is to iteratively retrieve and sparsify the inputs and outputs of the retrieval process, respectively. In the beginning, the visual modality of the generation cue is completely empty, but with each iteration, we provide the memory with more information. This, way the memory will gradually get closer to a pattern similar to those that it stores. Recall that cues with less active bits lead to retrieved vectors with more active bits. The process stops once the number of bits in the generation is in the predefined acceptance interval.

An overview of the architecture of the iterative generation method can be seen in Fig. 13(a). Additionally, an illustrative example is provided in Fig. 13(b). The first shows how the visual patterns evolve throughout the generation process, while the second monitors the number of active bits throughout the generation process. The results of the iterative generation method can be seen in Fig. 14.

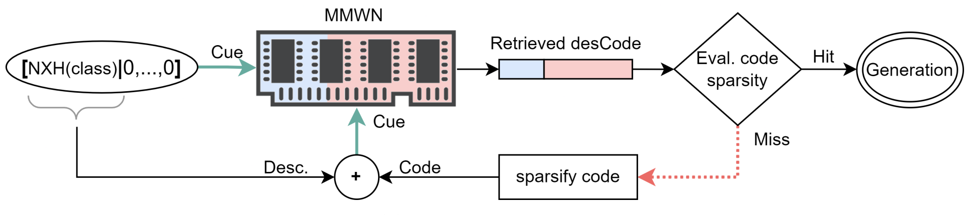
IV. Conclusion

In this final chapter, we will reiterate the motivation behind this work, and its main contributions and conclusions. Additionally, we reflect on the broader impact and future research ideas that this work enables.

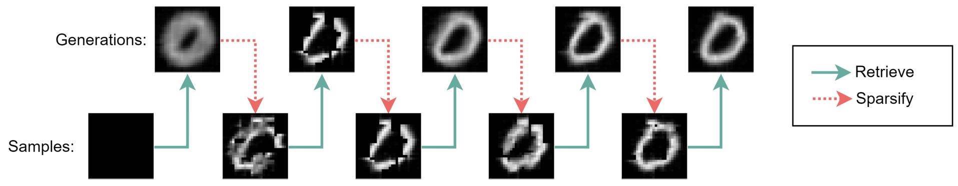
The Willshaw Network (WN) [17] is a simple Biologically-Inspired Neural Network that utilizes a Hebbian Learning Rule [3] to efficiently store large amounts of information. This model has not been used in practical applications since it requires its inputs to be Binary Sparse Distributed Representations (SDRs) [4], which do not occur naturally. A recent paper by Sá Couto and Wichert [1] proposed a prescription that transforms visual patterns into SDRs. This work opened up the opportunity for this work, where we intend to test the WN in a practical setting.

First, we extended the work of [1], by doing a thorough analysis of the storage and retrieval processes of MNIST patterns on the Willshaw Network. From this analysis, we noticed that the noise added by the memory on retrieval had some interesting properties, and we hypothesized that this noise might be beneficial in practice.

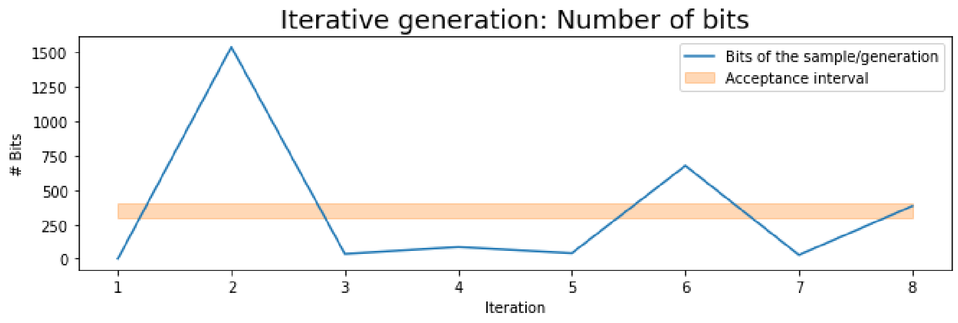
We then built a decoding module that allowed us to visually inspect the contents of the memory and confirm our hypothesis about the noise of retrieval. Experiments on noisy and noiseless cues were performed. Results showed that the memory exhibits great information-completion capabilities, but struggles with cues that have an excess of information. Additionally, we tried to use the memory and the decoder as a generative model but the results were not impressive.



(a) Method overview



(b) Iterative generation Example: Images



(c) Iterative generation Example: Sparsity

Fig. 13. **Iterative Generation: Three perspectives**. A Multi-modal Willshaw network is used to generate patterns. A description and an empty code are given to the memory as a retrieval cue. The memory returns a generation, and its sparsity is evaluated. If the sparsity of the generation is within a predefined interval, then the generation process ends; otherwise, the generation is "sparsified" (i.e. we delete bits from it until a certain level of activity is reached). The sparsified code is then fed back to the memory, and we repeat the process. **(a):** The process begins by providing the multi-modal memory with a desCode (DC) where the description is a new noisy-X-Hot (NXH) encoding of the class we want to generate from, and the code is empty (all zeros). Then we take the code from the retrieved desCode and evaluate its sparsity. If the sparsity is a "hit", the process ends here. Otherwise, we "sparsify" the retrieved code, join it with the original description, and repeat the process. **(b):** The patterns on the bottom row are retrieval cues that act as "samples" in the generation process. The memory retrieves (green arrows) these samples and produces generations which are displayed on the top row. If the generation's sparsity is outside the acceptance interval, we randomly delete bits from the generated code (dashed red arrows) to obtain a new sample. **(c):** Here we plot the number of bits throughout the iterative generation process. This graph corresponds to the same example as Fig. 13(b). The first sample has zero active bits since it corresponds to an empty code. The sparsity of the encoding process will oscillate (blue line): When we "sparsify" we bring the number of bits down. On retrieval, the memory will complete the missing information and add more bits. When the output of the memory falls within the acceptance interval (orange) the generation process ends.

Finally, we proposed the Multi-Modal Willshaw Network: a new way to use the memory which serves as a framework for practical applications. To test this framework, we experimented on the MNIST dataset. Not only did we improve the memory's retrieval performance, but we also successfully enabled new applications such as classification and generation.

Our results highlight the flexibility of the Multi-Modal framework to perform various tasks. Moving forward, here are some recommendations for future work:

- While the retrieval, reconstruction, generation and classification results were good, there is still plenty of room for improvement. A thorough analysis where multiple parameter configurations are tested will surely improve the results reported here.
- The iterative retrieval method proposed here can be modified and applied to other tasks. Some suggested changes include: (a) Sparsifying dense vectors with the memory itself; (b) monitoring the distribution as an acceptance measurement; and (c) monitoring the properties of multiple modalities simultaneously to determine the end of the iterative process.
- With the addition of multiple modalities in the Willshaw Network, it might be useful to experiment with new retrieval thresholding strategies: For instance, having one threshold for each modality.
- An analysis of the robustness of the model to demonstrate how this model does not suffer from the *Grandmother cell* problem: perform tasks with the Willshaw Network where neurons randomly "die" (become permanently inactive).
- Demonstrate the ability of this model to work in different domains with the same architecture. For instance use both the MNIST [6] and Fashion MNIST [42], or both CIFAR-10 and CIFAR-100 [43] simultaneously.
- While this work focuses on a Willshaw Network with two modalities, the idea of Multiple Modalities can be used in any Associative Memory Model (such as the Hopfield Network [12] and the Restricted Boltzmann Machine [44]) with any

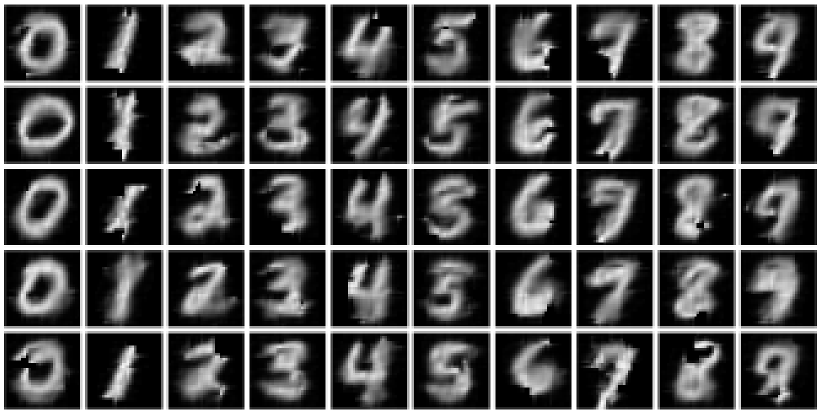


Fig. 14. **Iterative Generation with a MMWN: Generation Examples**. The generation quality of this approach is better than all the other strategies presented in this document. Almost all generations look like examples from the class that they belong to. Furthermore, there is some variance between generations of the same class. These results were obtained with an acceptance interval of $[400, 500]$ bits.

number of modalities. Building flexible software that allows us to continue testing this idea with a vast range of Models and Domains is crucial.

- To use Multi-Modal associative memories, encoding and decoding prescriptions that satisfy the constraints of the model are required. Building several of these modules is essential to test the idea of Multi-Modal Memories on a larger scale.

The biological memory is equipped with an impressive compression algorithm that can store the essential, and then infer the details to match the perception. Associative Memories are artificial models that aim to mimic these mechanisms of their biological counterpart. While grounded in biology and theoretically sound, these models have failed to deliver on practical applications. In this work, we proposed the simple idea of Multiple Modalities and applied it to one of the simplest Associative Memories. With the limited time available for a Master's Degree project we were already able to successfully perform many practical applications successfully. This work serves as a proof-of-concept for the idea of Multi-Modal Associative Memories as a framework to solve Artificial Intelligence Tasks. With further research efforts, this idea can be experimented on different models and modalities, and hopefully, get us closer to full-fill the long-standing promises of Associative Memories.

References

[1] L. Sa-Couto and A. Wichert, “Storing object-dependent sparse codes in a willshaw associative network,” *Neural Computation*, vol. 32, no. 1, pp. 136–152, 2020.
[2] K. Daniel, “Thinking, fast and slow,” 2017.
[3] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.
[4] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin, “Biological and machine intelligence (bami),” 2016, initial online release 0.4.
[5] B. A. Olshausen and D. J. Field, “Emergence of simple-cell receptive field properties by learning a sparse code for natural images,” *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.
[6] Y. LeCun, “The mnist database of handwritten digits,” *http://yann. lecun. com/exdb/mnist/*, 1998.
[7] G. Palm, *Chapter XII How Useful are Associative Memories?*, ser. North-Holland Mathematics Studies, L. Ricciardi and A. Scott, Eds. North-Holland, 1982, vol. 58.
[8] ——, “On associative memory,” *Biological cybernetics*, vol. 36, no. 1, pp. 19–31, 1980.
[9] T. Kohonen, *Self-organization and associative memory*. Springer Science & Business Media, 2012, vol. 8.
[10] A. M. Wichert, *Principles of Quantum Artificial Intelligence: Quantum Problem Solving and Machine Learning*. World scientific, 2020.
[11] T. Kohonen, P. Lehtiö, J. Rovamo, J. Hyvärinen, K. Bry, and L. Vainio, “A principle of neural associative memory,” *Neuroscience*, vol. 2, no. 6, pp. 1065–1076, 1977.
[12] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.
[13] J. A. Anderson, “A memory storage model utilizing spatial correlation functions,” *Kybernetik*, vol. 5, no. 3, pp. 113–119, 1968.
[14] T. Kohonen, “Correlation matrix memories,” *IEEE transactions on computers*, vol. 100, no. 4, pp. 353–359, 1972.
[15] J. A. Anderson, “A simple neural network generating an interactive memory,” *Mathematical biosciences*, vol. 14, no. 3-4, pp. 197–220, 1972.
[16] K. Nakano, “Associatron-a model of associative memory,” *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 380–388, 1972.
[17] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, “Non-holographic associative memory,” *Nature*, vol. 222, no. 5197, pp. 960–962, 1969.
[18] K. Steinbuch, “Die lernmatrix,” *Kybernetik*, vol. 1, no. 1, pp. 36–45, 1961.
[19] ——, “Automat und mensch,” in *Automat und Mensch*. Springer, 1965, pp. 390–411.
[20] G. Palm, *Neural assemblies: An alternative approach to artificial intelligence*. Springer Science & Business Media, 1982, vol. 7.
[21] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
[22] S. B. Laughlin and T. J. Sejnowski, “Communication in neuronal networks,” *Science*, vol. 301, no. 5641, pp. 1870–1874, 2003.
[23] P. Lennie, “The cost of cortical computation,” *Current biology*, vol. 13, no. 6, pp. 493–497, 2003.
[24] G. Palm, “On the information storage capacity of local learning rules,” *Neural Computation*, vol. 4, no. 5, pp. 703–711, 1992.
[25] F. Gunther Palm, F. T. Sommer, and A. Strey, “Neural associative memories,” *Associative Processing and Processors*, pp. 307–326, 1997.
[26] A. Newell and H. A. Simon, “Computer science as empirical inquiry: Symbols and search,” in *ACM Turing award lectures*, 2007, p. 1975.
[27] T. M. Mitchell *et al.*, “Machine learning,” 1997.
[28] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.
[29] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, “Deep blue,” *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.
[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[31] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision*. Cengage Learning, 2014.
[32] D. Marr, “Vision: A computational investigation into the human representation and processing of visual information,” 1982.
[33] K. Fukushima, “Neocognitron: A hierarchical neural network capable of visual pattern recognition,” *Neural networks*, vol. 1, no. 2, pp. 119–130, 1988.
[34] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.
[35] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
[36] Y. LeCun, Y. Bengio *et al.*, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.
[37] S. Lloyd, “Least squares quantization in pcm,” *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.
[38] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip *et al.*, “Top 10 algorithms in data mining,” *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.
[39] M. V. Chafee, B. B. Averbeck, and D. A. Crowe, “Representing spatial relationships in posterior parietal cortex: single neurons code object-referenced position,” *Cerebral Cortex*, vol. 17, no. 12, pp. 2914–2932, 2007.
[40] D. J. Felleman and D. C. Van Essen, “Distributed hierarchical processing in the primate cerebral cortex.” *Cerebral cortex (New York, NY: 1991)*, vol. 1, no. 1, pp. 1–47, 1991.
[41] J. Hawkins and S. Blakeslee, *On intelligence*. Macmillan, 2004.
[42] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
[43] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” 2009.
[44] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines,” *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.