A NEOCORTEX-BASED MODEL FOR PATTERN RECOGNITION

by

Ollantay Medina Huamán

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
in
COMPUTING AND INFORMATION SCIENCES AND ENGINEERING

UNIVERSITY OF PUERTO RICO

MAYAGÜEZ CAMPUS

2014

Approved by:

_____	_____
Edgar Acuña, PhD Member, Graduate Committee	Date
_____	_____
Domingo Rodríguez, PhD Member, Graduate Committee	Date
_____	_____
Alexander Urintsev, PhD Member, Graduate Committee	Date
_____	_____
Vidya Manian, PhD President, Graduate Committee	Date
_____	_____
Esov Velázquez, PhD Representative of Graduate Studies	Date
_____	_____
Wilson Rivera, PhD CISE PhD Program Coordinator	Date

ABSTRACT

The information society needs to process massive volumes of data in automated ways. The data is more meaningful if it is considered as sequences of patterns rather than isolated patterns. A trained human brain has outstanding capabilities for pattern recognition tasks, which could be very advantageous for automated pattern recognition systems.

This thesis proposes a generic pattern recognition model. This model is mainly based on the known operation of the neocortex and is oriented to deal with sequences of patterns or data streams.

The theoretical part of the model is formally stated and an implementation is outlined. The theoretical model also establishes a more general framework for treatment of space time data through a dimensionality reduction process. For a given instance of space time data, the process characterizes a space time region that might be called an invariant semantic representation.

The model exhibits desirable properties for a pattern recognition system, such as spatial and temporal autoassociativity, spatial and temporal noise tolerance, recognition under sequence contextualization, and input prediction.

RESUMEN

La sociedad de la información necesita procesar volúmenes masivos de datos de forma automatizada. Estos datos son más significativos si son considerados como secuencias de patrones en vez de datos aislados. Un cerebro humano entrenado tiene capacidades sobresalientes para tareas de reconocimiento de patrones, lo cual podría ser muy ventajoso para sistemas automatizados de reconocimiento de patrones.

Esta tesis propone un modelo de reconocimiento de patrones genérico. Este modelo está basado fuertemente en la operación conocida de la neocorteza y está orientado a tratar con secuencias de patrones o flujos de datos.

La parte teórica del modelo es planteada formalmente y una implementación es esbozada. El modelo teórico también establece un marco más general para tratar datos de espacio tiempo a través de un proceso de reducción de dimensionalidad. Para una instancia dada de datos de espacio tiempo, el proceso caracteriza una región de espacio tiempo que podría ser llamada una representación semántica invariante.

El modelo exhibe propiedades deseables para un sistema de reconocimiento de patrones, tales como autoasociatividad espacial y temporal, tolerancia al ruido espacial y temporal, reconocimiento bajo contextualización de una secuencia, y predicción de entrada.

To Karen Sofia

ACKNOWLEDGEMENTS

It is with sincere gratitude that I want to acknowledge the support and help of Dr. Vidya Manian. Thank you for your advice, time, and motivation.

I also want to thank to my committee members Dr. Edgar Acuña, Dr. Domingo Rodríguez and Dr. Alexander Urintsev for their time and support.

Additionally, I would like to thank the staff of the doctoral program in Computer and Information Sciences and Engineering of the University of Puerto Rico and also the staff of the Laboratory for Applied Remote Sensing and Image Processing.

A special gratitude and love goes to my family for their unconditional support and love.

TABLE OF CONTENTS

TABLE LIST

Tables **Page**

FIGURE LIST

Figures **Page**

1 INTRODUCTION

Humans are well known for their remarkable abilities to recognize patterns; the generalization and invariance capabilities of the human brain enable us to recognize successfully new patterns just after exposure to few new samples.

The human brain has a lot in common with the rest of the mammalian brains, which in turn have an almost unique feature compared to other species, the outermost layer known as neocortex; this part is believed to be the source of the superior intelligence of mammals enabling the ability to deal with complex patterns of information and to do so in a hierarchical fashion [1]. Humans, in proportion, have one of the biggest neocortex among mammals. The progress in neurosciences has revealed a lot of facts about the structure and operation of the neocortex [2] and [3]; the current knowledge is good enough to formulate computational models that could capture the abilities to recognize patterns successfully [4], [5] and [6]. The appealing characteristics that this kind of model is expected to capture are: Handling of space and space-time patterns, noise resistance, fault-tolerance and real-time response. Other previous attempts to produce pattern recognition models based on the neocortex or neural networks are: Neocognitron [7], HMAX [8], and Deep Learning models [9].

In our times, the information society needs to process massive volumes of data, the pattern recognition task is ubiquitous; humans can no longer extract information from this data without computational aid [10] and [11]. The need for new pattern recognition models with characteristics like the ones mentioned for a neocortex-based model is undeniable.

The understanding of human brain and human intelligence is a formidable task that could bring a new class of intelligent machines capable of extracting information from the colossal volume of data generated at every moment.

1.1 Motivation

This work aims to develop a neocortex-based model and use it for pattern recognition tasks. The pattern recognition domain in our case is restricted to the supervised classification problem; the model should be suitable, but not restricted, to identifying space and space-time patterns in images.

1.2 Problem Statement

Provide a formal model and an implementation of a classifier. This classifier should exploit the characteristic features of the human neocortex in order to assist the pattern recognition task of complex spatio-temporal patterns.

To overcome the challenges, the model has to be based on recent findings about the structure and operation of the neocortex, capturing the essence of its functionality.

1.3 Contributions

The primary contribution of this thesis is the development of a robust model for classification capable of:

- Spatial and temporal autoassociativity.
- Spatial and temporal noise tolerance.
- Recognition under sequence contextualization.
- Input prediction.

Secondary contributions are improvements in the understanding of:

- How the brain works.
- What the neocortex might store.
- How abstractions arise in the brain.
- How the brain stores and retrieves information.

The secondary contributions are a consequence of the proposed model from a high level functional point of view. This model, as well as other similar models, provide a partial functional perspective on how the neural networks in the brain, learn and store (complex) information. The coding scheme proposed for the model is motivated by neurological

evidence on how individual neurons in the neocortex process a specific information subdomain. At running time, an instance of input data learned by the model produces an internal representation that might be considered the equivalent of an abstraction in the brain. Finally, input data triggers a sequential process in the model that learns and stores the incoming pattern and prepares retrieval of the next pattern by means of predictions.

1.4 Summary of Following Chapters

Chapter 2 introduces the necessary theoretical background for this study. Chapter 3 deals with the neocortex-based model and model implementation. Chapter 4 presents experiments and data analysis. In Chapter 5 conclusions are presented and future work is proposed. Chapter 6 presents the ethical considerations relative to this thesis.

2 THEORETICAL BACKGROUND

There are two important sections in this chapter, and. The first section, neuroscience concepts, gives a brief physiological overview about the main components of the neocortex and how learning occurs in it. The second section on mathematical and computational concepts, presents relevant concepts to establish a mathematical formulation and to propose computational operation. In particular, Section 2.2.2 presents a brief explanation on what coding schemes to store information are known. Section 2.2.3 presents a view on how the brain might store information. Sections 2.2.5 and 2.2.6 present a framework that attempts to explain how the neocortex handles incoming and stored information. Section 2.2.7 introduces a formal model for the input data. Finally, Section 2.2.8 shows how the model can handle input data once it is transformed into the internal format of the model.

2.1 Neuroscience Concepts

2.1.1 Neuron

A cell with two kinds of branches: dendrites and an axon. The neuron receives input signals from other neurons, integrates them, and generates its own signal, an electric impulse

that travels along the axon away from the cell body. The axon makes contact with dendrites and cell bodies of other neurons; thus, the output signal of one neuron becomes input to other neurons. The structure that allows a neuron to pass an electrical or chemical signal to another cell is called synapse.

2.1.2 Synapses

The importance of a synapse to the firing of a neuron is called the synaptic weight. There are two kinds of synapses: excitatory and inhibitory. Excitatory synapses help a neuron to fire, inhibitory synapses hinder firing. An entire neuron can be considered as excitatory or inhibitory according to the kind of synapses that its axon makes. Finally, the firing of the neuron depends on the neuron's threshold, meaning that the combined input of excitatory and inhibitory impulses must surpass the threshold to trigger the firing.

2.1.3 Dendrites

The current knowledge shows that dendrites are more than just conduits to bring inputs of the cell body. Dendrites can be considered as complex non-linear processing elements in themselves [12]. Two types of dendrites can be distinguished:

- *Proximal Dendrites*: The dendrite branches closest to the cell body. Multiple active synapses on proximal dendrites have a roughly linear additive effect at the cell body.

- *Distal Dendrites*: The dendrite branches farther from the cell body. Distal dendrites are thinner than proximal dendrites. They connect to other dendrites at branches in the dendritic tree and do not connect directly to the cell body. Distal dendrites act as semi-independent processing regions, if enough distal synapses become active at the same time within a short distance along the dendrite, they can generate a dendritic spike that can travel to the cell body.

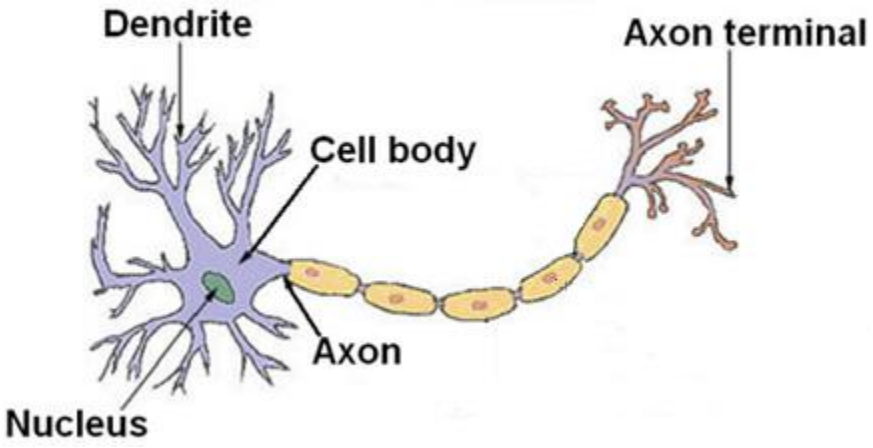

Figure 2.1 Structure of a typical neuron

2.1.4 Neocortex

The neocortex is the outermost part of the brain of mammals; it is a thin sheet of 6 layers of neurons. Humans have in proportion the biggest neocortex among mammals. Neuroscientist V. Mountcastle discovered the columnar organization of the neocortex, which is very uniform across its surface [13]. Neurons in a column receive common inputs, are

interconnected and have common outputs. Mountcastle argues that these columns may constitute a fundamental computational unit in the neocortex and they should work under a common cortical algorithm [14]. Further research discovered that the visual cortex possess a hierarchical organization [1].

2.1.5 Hebbian Learning

Brains are networks of neurons connected through synapses, there should be a mechanism that gives them the ability to learn. This idea led to the theory that learning could be the consequence of changes in the strengths of the synapses. The best-known theory of learning based on synaptic plasticity is that proposed by Hebb [15], who postulated that connection strengths between neurons are modified based on neural activities in the presynaptic and postsynaptic cells:

When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

2.2 Mathematical and Computational Concepts

2.2.1 Neuron Model

The formal neuron model considers that a neuron in a net operates in synchronous time steps. Within each step a neuron can fire at most once, and the firing during interval t depends only on the neuron's inputs during interval $t-1$ and the threshold. The output is 1 when the neuron fires and 0 otherwise. An n-input neuron is modeled by a linear threshold function [16].

$$F: \{0,1\}^n \rightarrow \{0,1\}$$

Let $x_{i,t} \in \{0,1\}$ be the ith input at time t, and let w_i be the weight of the ith input, the weighted sum of the inputs is defined by

$$S_t = \sum_{i=0}^{n-1} w_i x_{i,t}$$

Let F_t be the output at time t, and let c be the threshold, the output is given by

$$F_t = \begin{cases} 1 \text{ if } S_{t-1} \geq c \\ 0 \text{ otherwise,} \end{cases}$$

For a neuron to fire, the weighted sum must reach or exceed the threshold. It should be noted that the recovery time, that a neuron needs in order to fire again, varies from neuron to neuron, the following could be a better description of recovery after firing: Firing raises

the neuron's threshold to a value greater than the maximum weighted sum, and the neuron cannot fire again immediately; thereafter, the threshold drops until firing becomes possible.

2.2.2 Local, Dense and Sparse codes

An important aspect of an information processing system is the way in which information is represented. A neural system like the brain, with a set of *N* binary neurons, could use one of the following coding schemes [17]:

- *Local codes*, in which each item is represented by a separate neuron. A neuron in this scheme is called a *grandmother cell*. This scheme is simple and easy to decode. However, it has a very low representational capacity (*N* items at most), and it is not fault tolerant (a lost cell means a lost item).
- *Dense codes*, in which each item is represented by the combination of activities of all neurons. This scheme has a very high representational capacity (2^N items). However, coding and decoding is difficult since all neurons are involved in the process, and requires redundancy schemes (at the expense of capacity) to make it fault tolerant; otherwise, a lost cell renders the code useless.
- *Sparse codes*, in which each item is represented by a small set of neurons. This scheme combines advantages of local and dense codes while avoiding most of their

drawbacks. It is still easy to decode, has a sufficiently high representational capacity, and is fault tolerant.

2.2.3 Sparse Coding in the Brain

The spatial receptive fields of simple cells in mammalian cortex can be characterized as being *localized*, *oriented*, and *bandpass*, comparable with the basis functions of wavelet transforms. These receptive field properties seem to produce, in terms of a strategy, a sparse distribution of output activity in response to natural images [18]. Field [19] applied this idea to simple cells in primary visual cortex suggesting that basis functions limited both in space and frequency maximize sparseness when applied to natural images. Földiak [20] proposed an algorithm to find sparse encodings in neural network models based on anti-Hebbian lateral connections within a layer of nonlinear artificial neurons and Hebbian forward connections between these layers, combined with a local threshold control mechanism to learn a sparse code with low information loss, representing the inputs by the combination of uncorrelated components.

2.2.4 Sparse Representations

The sparse representation of a signal $y \in \mathbb{R}^N$ in a given overcomplete dictionary A, can be stated as $y = Ax$, where A is a $N \times M$ matrix (with $M > N$) containing the elements of an overcomplete dictionary in its columns, x is a $M \times 1$ coefficient vector, whose $\|x\|_0$ is

minimal [21]. Under these considerations, the problem of sparse representation is to find x such that:

$$x = min_{x'} \|x'\|_0 \qquad \text{s.t.} \qquad y = Ax$$

where $\|x\|_0$ is the l_0 norm and is equivalent to the number of non-zero components in vector x. Vector x is called the sparse code of y.

2.2.5 Memory-Prediction Framework

The memory-prediction framework is a theory of brain function based mainly on the role of the mammalian neocortex in matching sensory inputs to stored memory patterns and how this process leads to predictions of what will happen in the near future. The theory postulates that the remarkably uniform morphology of cortical tissue reflects a common single algorithm which underlies all cortical information processing [5].

The central idea of the theory is based on a hierarchy of recognition where bottom-up inputs interact with top-down expectations to generate predictions of subsequent expected inputs. Each hierarchy level remembers frequently observed temporal sequences of input patterns and generates *labels* for these sequences that are propagated up the hierarchy. As one moves up the hierarchy, representations gain:

- *Extent*, bigger areas of the sensory domain are covered.
- *Temporal stability*, higher-level *labels* tend to be more stable than lower-level *labels*.

- *Abstraction*, successive extraction of features, improves invariant representations, which in higher levels allow recognition of increasingly abstract entities.

2.2.6 Hierarchical Temporal Memory

Hierarchical temporal memory (HTM) is a model that attempts to implement the memory-prediction framework. Since its conception, HTM has evolved in several aspects, as can be seen in [22], [23] and [24]. The current known version of HTM has a strong influence in the model proposed in this study.

HTM is fundamentally a memory-based system with a hierarchical organization, trained on lots of time-varying data. HTM can be viewed as a new form of neural network with specific architectural guidelines. HTM uses a neuron model more complex than the classical one; these neurons are arranged in columns, layers, regions, and hierarchies. The way data is stored and accessed is logically different from the standard model used in classic computer memory; the standard model has a flat organization and does not have an inherent notion of time. A programmer can implement any kind of data organization and structure on top of the flat computer memory, and has control over how and where information is stored. By contrast, HTM memory is more restrictive, it controls where and how information is stored, following a distributed fashion. The main characteristics of the model are:

- *Hierarchy*: An HTM network consists of regions arranged in a hierarchy. The region is the main unit of memory and prediction in an HTM. Typically, each HTM region represents one level in the hierarchy. Ascending the hierarchy always leads to convergence; multiple elements in a child region converge onto an element in a parent region. However, due to feedback connections, information also diverges as you descend the hierarchy. The benefit of hierarchical organization is efficiency. It significantly reduces training time and memory usage because patterns learned at each level of the hierarchy are reused when combined in novel ways at higher levels. Sharing representations in a hierarchy also leads to generalization of expected behavior. The hierarchy enables a new object in the world to inherit the known properties of its sub-components.
- *Regions*: The notion of regions wired in a hierarchy comes from biology. Biologists divide the neocortex into different areas or "regions" primarily based on how the regions connect to each other. Some regions receive input directly from the senses and other regions receive input only after it has passed through several other regions. It is the region-to-region connectivity that defines the hierarchy. All neocortical regions look similar in their details. They vary in size and where they are in the hierarchy, but otherwise they are similar.

- *Sparse distributed representations*: Although neurons in the neocortex are highly interconnected, inhibitory neurons guarantee that only a small percentage of the neurons are active at one time. Thus, information in the brain is always represented by a small percentage of active neurons within a large population of neurons. This kind of encoding is called a “sparse distributed representation”. A single active neuron conveys some meaning but it must be interpreted within the context of a population of active neurons to convey the full meaning. The representation is obtained in a "winner takes all" fashion, the first cell that activates in a zone, inhibits neighbor cells in the zone. It may seem that this process generates a large loss of information as the number of possible input patterns is much greater than the number of possible representations in the region. However, both numbers are really big. The actual inputs seen by a region will be a miniscule fraction of all possible inputs. The theoretical loss of information will not have a practical effect.
- *Learning*: An HTM region learns about its world by finding patterns and then sequences of patterns in sensory data. The region does not “know” what its inputs represent; it works in a purely statistical realm. It looks for spatial patterns, combinations of input bits that occur together often. It then looks for temporal patterns or sequences, how these spatial patterns appear in sequence over time. After initial training, an HTM can continue to learn or, alternatively, learning can be

disabled after the training phase. Once an HTM has learned the basic statistical structure of its world, most new learning occurs in the upper levels of the hierarchy.

- *Prediction*: Every region of an HTM stores sequences of patterns. By matching stored sequences with current input, a region forms a prediction about what inputs will likely arrive next. HTM regions actually store transitions between sparse distributed representations. An HTM region will make different predictions based on context that might stretch back far in time. The majority of memory in an HTM is dedicated to sequence memory, or storing transitions between spatial patterns. Some key properties of HTM prediction are:
    - Prediction is continuous. As the inputs come in, HTM is constantly predicting what will happen next.
    - Prediction occurs in every region at every level of the hierarchy.
    - Predictions are context sensitive. Predictions are based on what has occurred in the past, as well as what is occurring now. Thus an input will produce different predictions based on previous context. An HTM region learns to use as much prior context as needed, and can keep the context over both short and long stretches of time, ability known as "variable order" memory.
    - Prediction leads to stability. The output of a region, its prediction, becomes more stable the higher they are in the hierarchy.

- A prediction tells us if a new input is expected or unexpected. Each HTM region is a novelty detector. Because each region predicts what will occur next, it "knows" when something unexpected happens.
- Prediction helps make the system more robust to noise. When an HTM predicts what is likely to happen next, the prediction can bias the system toward inferring what it predicted, avoiding the effects of noise.

2.2.7 Dynamical Systems

A dynamical system is a tuple (T, S, R), with T a set of non-negative times, S a state space, and R an evolution function, $R: T \times S \rightarrow S$. The coordinates in the state space give a complete description of the system. Given a current state of the system, the evolution function predicts the next state or states. These predictions implicate the concept of time, which may be discrete or continuous. The evolution function is deterministic if each state has a unique consequent, and is stochastic if there is a probability distribution of possible consequents for a given state. The evolution function can be represented by a differential equation or a difference equation [25].

2.2.8 Deterministic Finite Automata

A deterministic finite automaton (DFA) is a computational model defined as a 5-tuple [26]

$$(Q, \Sigma, \delta, q_0, F)$$

where:

Q is a finite set called the states,

Σ is a nonempty finite set called the alphabet,

$\delta: Q \times \Sigma \rightarrow Q$ is the transition function,

$q_0 \in Q$ is the start state, and

$F \subseteq Q$ is the set of accept states.

The members of the alphabet are regarded as symbols. A string over an alphabet is an ordered finite sequence of symbols taken from the alphabet. A DFA either accepts or rejects a string produced over the alphabet of the DFA. The language of a DFA M is the set of all strings that M accepts, denoted as $lang(M)$.

2.2.9 Pattern Recognition

Pattern recognition is the study of how machines can observe the environment, learn to distinguish patterns of interest from their background, and make sound and reasonable decisions about the categories of the patterns [27].

Watanabe [28] defines a pattern "as opposite of a chaos; it is an entity, vaguely defined, that could be given a name", and establishes that the recognition/classification of a given pattern may be achieved by one of the following tasks:

- Supervised classification, in this case the pattern is identified as a member of a predefined class.
- Unsupervised classification, in this case the pattern is assigned to a, until now, unknown class.

3 A NEOCORTEX-BASED MODEL

The accepted view among evolutionary biologists and cognitive neuroscientist is that evolution molded the human brain to solve problems related to surviving in an unstable outdoor environment and to do so in nearly constant motion [29]. Solving survival problems requires building and storing a model of the real world, task mainly done by the neocortex. The brain uses this memory-based model to make continuous predictions of future events; which would define intelligence as the ability to successfully predict the future, far enough to be of real use to the survival of an organism [5]. The real world model built by the brain, at first, is just a suitable representation for survival purposes. The stored representation is not a comprehensive representation of the real world, but a sample from a limited domain, dictated by the resolution of our natural sensors, the senses [30].

The tenets of HTM are analyzed and restated as a neocortex-based model; this model proposes a generalization of the role of sparse distributed representations by using a set of mappings, a semantic representation and a set of codebooks. Also, the model has a clearer and more complete formal specification of the hierarchy than HTM in [24]. In this model, the handling of input prediction is clearer and well defined using the transition function of a DFA. Finally, the general way in which the model uses codebooks, gives freedom to implement them, using any method that can group data and obtain representatives from these groups.

The model uses the concepts introduced in Chapter 2 in the following way. The input data can be formally treated as a sequence of states generated by a dynamical system. The model transforms this sequence into a semantic representation using a mapping function and a local code. The semantic representation of a single state is a high-dimensional Boolean vector. Two vectors have semantic similarity if they have common components. Once the input is in semantic form, as a sequence of Boolean vectors, the concept of codebooks from vector quantization is used to model archetypes that represent a (sparse) group of characteristics. The codebook is a finite set that can be considered as a set of symbols, an alphabet; under this consideration, the input sequence is equivalent to a string over an alphabet. The training dataset in a supervised classification problem is finite, making it comparable to a finite set of strings. The simplest computational model that can process a finite set of strings, a finite language, is a DFA. In the model, DFA are used to handle learning, acceptance (or rejection), and prediction of incoming patterns (strings). Also, the model aims to perform a dimensionality reduction of space and time using hierarchies and grouping sequences of transformed patterns.

The model deals effectively with classification problems both in supervised manner and unsupervised manner. This study focuses only in the supervised version of the problem, which is stated as follows.

The problem formulation considers an extended version of the formal supervised pattern recognition problem, suitable to handle data streams. The theoretical model and model implementation proposed are capable of solving this problem, and are based mainly in the current knowledge of how the neocortex works.

3.1 Problem Formulation

3.1.1 Supervised Pattern Recognition Problem

Given:

- A set of instances X and a set of labels (classes) Y.
- An unknown function $g: X \rightarrow Y$ that maps input instances $x \in X$ to output labels $y \in Y$.
- A training set $E = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, assumed to represent accurate examples of the mapping g.

Produce:

- A function $\bar{g}: X \rightarrow Y$ that approximates as closely as possible the mapping g.

3.1.2 Extended Supervised Pattern Recognition Problem

The previous formulation is extended in the definition of the set X, that now incorporates sequences:

- A set of instances X, where $x \in X$ is a sequence of states generated by an unknown dynamical system $D = (T, S, R)$, with T a non-negative time interval, S the state

space, and R the evolution function defined as $R: T \times S \to S$. If D is a real dynamical system, T is defined as the non-negative reals; otherwise, if D is a discrete dynamical system, T is defined as the non-negative integers. In both cases S is a Banach space over the vector space $\mathbb{R}^n$.

The goal of this formulation is the same as the previous one, it produces a function $\bar{g}$ that approximates g.

3.2 Theoretical Model

The idea behind this model is to solve the extended supervised pattern recognition problem by constructing an automaton A capable of computing $\bar{g}$. The following assumptions are made:

- $\bar{g}$ is a computable function.
- For the extended set of instances X:
    - The principle of locality, in the spatial and temporal sense, holds. Instances that exhibit spatial proximity or temporal proximity can be considered *similar* under some metric.
    - For all $x \in X$, x is a finite sequence generated by an unknown discrete dynamical system.

3.2.1 The Automaton A

Let U be a set of unknown discrete dynamical systems that generates X:

$$U = \{D_i : D_i = (T_i, S_i, R_i)\}$$

where T_i is a finite set restricted to the positive integers, S_i is the state space of D_i and R_i is the evolution function $R_i : T \times S_i \to S_i$. Assume, without loss of generality, that every D_i can use the state space $S = \bigcup_{S_i \in U} S_i$, transforming U into:

$$U = \{D_i : D_i = (T_i, S, R_i)\}$$

where $R_i : T_i \times S \to S$.

Let $lang(U)$ be the set of all possible sequences generated by U, more specifically:

$$lang(U) = \{x : (\exists D_i)x = (s_k)_{k=1}^{|T_i|}, s_k \in S, s_k = R_i(s_{k-1}), s(1) = s_1\}$$

Hence $X \subseteq lang(U)$.

Let A be an automaton defined as:

$$A = (H, M)$$

where $H = (h_1, h_2, \cdots, h_{n+1})$ is a tuple of mappings and $M = (M_1, M_2, \cdots, M_n)$ is a tuple of deterministic finite automata. The automaton A can be thought as a hierarchy of deterministic finite automata that uses mappings in every level to preprocess and to transform its input sequence. An input sequence x is said to be accepted by A if every automaton in the hierarchy accepts the respective transformation of x. $lang(A)$ is the set of all sequences that A accepts. Automaton A is used to recognize X in the sense that $X \subseteq lang(A)$. The construction of A is as follows:

- h_1 is defined as $h_1: S \to \Sigma_1$ such that $h_1(s) = a$ where $s \in S$ and $a \in \Sigma_1$. h_1 internally performs four mappings using a bounded domain $\Omega \subset S$ such that $dom_S(X) \subseteq \Omega$, where $dom_S(X) = \{s \in S: ((\exists x \in X)x = (s_i) \wedge s \in x)\}$. The mappings are as follows:
    - $h_{1,1}: \Omega \to \Lambda$, where Ω is a bounded set such that $dom_S(X) \subseteq \Omega \subset S$, and Λ is a lattice.
    - $h_{1,2}: \Lambda \to \{0,1\}^m$, where Λ is a lattice, $m = |\Lambda|$, and $\{0,1\}^m$ is a m-dimensional Boolean space.
    - $h_{1,3}: \{0,1\}^m \to \Gamma_1$, where $\{0,1\}^m$ is a m-dimensional Boolean space and $\Gamma_1 \in \{0,1\}^m$ is a codebook obtained by a Vector Quantization process in the Boolean space under the Hamming distance.
    - $h_{1,4}: \Gamma_1 \to \Sigma_1$, where $\Gamma_1 \in \{0,1\}^m$ is a codebook and Σ_1 is a finite set of symbols (an alphabet), this mapping is bijective.
    - $h_1(s) = h_{1,4}\left(h_{1,3}\left(h_{1,2}\left(h_{1,1}(s)\right)\right)\right) = a$ where $s \in S$ and $a \in \Sigma_1$.
- The notation h^i is a compact way to represent a transformation of a sequence over Ω into a string over Σ_i. $h^i: \Omega^* \to \Sigma_i^*$ computed as $h^i(s) = h_i\left(h_{i-1}\left(\dots h_2\left(h_1(s)\right)\dots\right)\right) = r$, where $s \in \Omega^*$ and $r \in \Sigma_i^*$.

- $h_i, 1 < i \leq n$ is defined as $h_i: \Sigma_i^t \rightarrow \Sigma_{i+1}$ such that $h_i(s) = b$, where $s \in \Sigma_i^t$ and $b \in \Sigma_{i+1}$. h_i computation requires choosing a fixed parameter t to group subsequences of length t in an unordered way. A goal of h_i is to reduce the dimensionality of the problem by imposing the constraint $|\Sigma_i| > |\Sigma_{i+1}|$. The computation is as follows:
    - $h_{i,1}: \Sigma_i \rightarrow \{0,1\}^m$, where $|\Sigma_i| = m$, such that $h_{i,1}(a_j) = b_j$ with $\Sigma_i = \{a_1, a_2, \cdots, a_m\}$ and $b_j \in \{0,1\}^m$ a canonical Boolean vector with a 1 in the *j*th position.
    - $h_{i,2}: \Sigma_i^t \rightarrow \{0,1\}^m$, where Σ_i^t represents the strings of length t over Σ_i and $\{0,1\}^m$ is a m-dimensional Boolean space, such that given a $s \in \Sigma_i^t$, $s = s_1 s_2 \ldots s_t$:

      $$h_{i,2}(s) = h_{i,2}(s_1 s_2 \ldots s_t) = h_{i,1}(s_1) \vee h_{i,1}(s_2) \vee \ldots \vee h_{i,1}(s_t) = r$$

      with $r \in \{0,1\}^m$.
    - $h_{i,3}: \{0,1\}^m \rightarrow \Gamma_i$, where $\{0,1\}^m$ is a m-dimensional Boolean space and $\Gamma_i \in \{0,1\}^m$ is a codebook obtained by a Vector Quantization process in the Boolean space under the Hamming distance.
    - $h_{i,4}: \Gamma_i \rightarrow \Sigma_{i+1}$, where $\Gamma_i \in \{0,1\}^m$ is a codebook and Σ_{i+1} is a finite set of symbols (an alphabet), this mapping is bijective.

- $h_i(s) = h_{i,4}\left(h_{i,3}\left(h_{i,2}(s)\right)\right) = b$ where $s \in \Sigma_i^t$ and $b \in \Sigma_{i+1}$.

- M_i is defined as $M_\mathrm{i} = (Q_\mathrm{i}, \Sigma_\mathrm{i}, \delta_\mathrm{i}, q_0, F_\mathrm{i})$, where if $p = |\Sigma_i|$ then:
  - $Q_i = \{q_0, q_1, \cdots, q_p\}$
  - $\Sigma_i = \{a_1, a_2, \cdots, a_p\}$
  - $\delta_i(q_j, a_k) = q_k$,

    if $((\exists x \in X, s \in \Sigma_i^*) h^i(x) = s \wedge a_j a_k$ is a substring of s)
  - q_0 is the initial state
  - $F_i = Q_i - \{q_0\}$
- Choose n and t in such a way that $h^n(x) = a$ with $x \in X$ and $a \in \Sigma_n$.
- h_{n+1} is defined as $h_{n+1}: \Sigma_n \to Y$ such that:

  $h_{n+1}(a) = y$, if $((\exists x \in X, a \in \Sigma_n, y \in Y)\ h^n(x) = a \wedge (x, y) \in E)$.

By construction, automaton A accepts, across its hierarchy, the sequences of X, thus $X \subseteq lang(A)$. In the ideal case that $X = lang(U)$, automaton A would be capable of recognizing the language of U, however due to the definition of H, automaton A is not guaranteed to be a decider of $lang(U)$; hence in the best case $lang(U) \subseteq lang(A)$.

The complete definition of H produces the mapping $h^{n+1}(x) = y$ built using the information of the training set E, hence h^{n+1} can be considered as a mapping that

approximates the unknown function g. Therefore $\bar{g} = h^{n+1}$ is a solution of the extended pattern recognition problem.

3.2.2 A simple example

The following example tries to show how automaton *A* processes a given sequence of patterns in order to accept or reject the sequence as part of its language. Working with a low-dimensional example is an oversimplification of the actual high-dimensional expected input; the process may seem counter-intuitive or flawed, but this example only aims to show the procedure that automaton *A* follows.

The example presented is a path over a space time plane, where space is the positive reals and time is also the positive reals as Figure 3.1 shown.

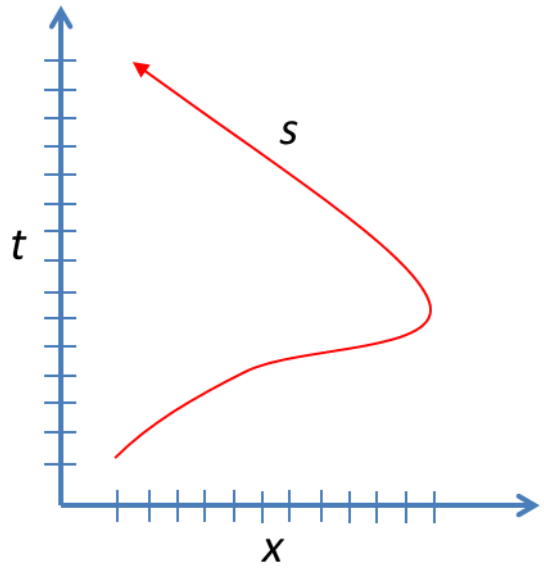

Figure 3.1 A simple sequence *s* over space time

The first step is to choose a domain $\Omega = [1,12]$ that contains all the states used by *s*. Time is mapped to the positive integers transforming *s* into a finite sequence. Then a lattice Λ transforms Ω into a finite set, leaving *s* also as a finite sequence as Figure 3.2 shown.

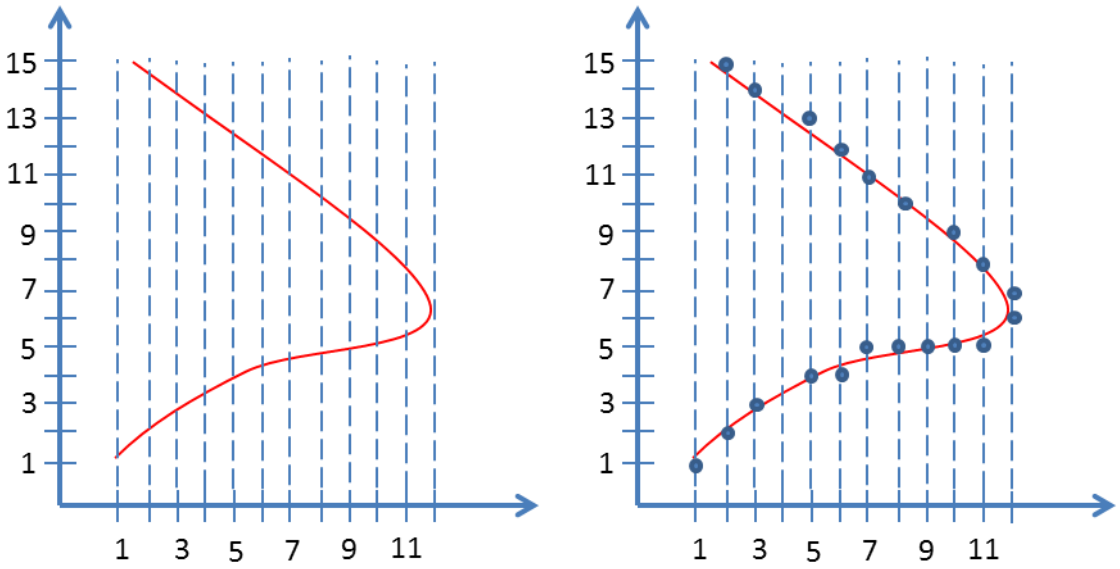

Figure 3.2 Discretization of space time that transforms *s* into a finite sequence

The next step creates a semantic representation of space by mapping the discrete space into a Boolean space; 12 elements in space imply a 12-dimensional Boolean space. The representation is semantic in the sense that if two vectors have an active bit in common, those vectors are using the same portion of space; or, the bit means the same in terms of space for both vectors. More bits in common indicate more similarity between vectors. The semantic representation over a Boolean space makes the Hamming distance a suitable metric to measure dissimilarity between vectors. Table 3.1 shows the resulting set of Boolean vectors for sequence *s*, where zeroes are omitted for clarity. The column *Alphabet* is explained with the help of Table 3.2 in the next paragraph.

Table 3.1 Boolean vectors representing sequence *s*

	Time Step	Space												Alphabet
		1	2	3	4	5	6	7	8	9	10	11	12	
sequence ***s***	**1**	1												a
	2		1											b
	3			1										c
	4					1	1							d
	5							1	1	1	1	1		e
	6												1	f
	7												1	f
	8											1		e
	9										1			e
	10								1					e
	11							1						e
	12						1							d
	13					1								d
	14			1										c
	15		1											b

After the first semantic representation of *s* is obtained, a vector quantization procedure follows; this procedure uses Hamming distance and a threshold to group vectors. As a result the numbers of different vectors to represent *s* is reduced. The resulting *codebook* is mapped to an alphabet which in turn serves to rewrite the sequence from Table 3.1 as *s*=`abcdeffeeeeddcb`. The codebook of Table 3.2, for the sake of the example is obtained

with a very loose threshold of 1 bit: two vectors are considered the same if they have at least 1 bit in common.

Table 3.2 First codebook and alphabet obtained from the Boolean set of vectors that represent *s* in Table 3.1

Space												Alphabet
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	
1												a
	1											b
		1										c
				1	1							d
						1	1	1	1	1		e
											1	f

At this point *s* is transformed into a string over an alphabet, here is where the deterministic finite automata *M* from automaton *A* come in use; their duty at every level of the hierarchy is to learn the needed transitions to accept *s* and other training data. After training, *M* should reject sequences that don't resemble the training data. Figure 3.3 shows the new space time representation of *s* using the first alphabet.

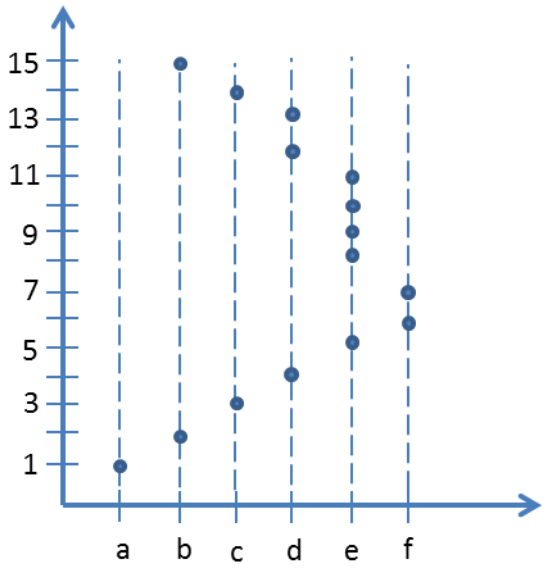

Figure 3.3 Space time representation of *s* using the first alphabet.

The previous codebook reduced the space state by grouping semantically-similar vectors; the procedure that follows is an attempt to compress time by taking close-in-time subsequences of an arbitrary length. This process maps the input into a more abstract time-dependent space where the set of states are fragments that represent change across time. To produce this mapping, time is considered in an unordered way; the temporal grouping parameter should not be too small, the purpose is to reduce the probability that two sequences, that should be considered different, are one the permutation of the other.

For the current example the temporal parameter is set to 3. The number of vectors in the last codebook is 6, which creates a new 6-dimensional Boolean space, where sequence *s* is represented by 5 vectors built upon subsequences of length 3, these results appear in Table

3.3. The new codebook obtained in Table 3.4 comes after applying vector quantization on vectors from Table 3.3; this procedure returns the alphabet seen in Table 3.4 and sequence *s* now is represented as *s*=ghhhg.

Table 3.3 Boolean vectors representing sequence *s* after applying a temporal grouping parameter of 3 time steps.

	Time Step	Space						Alphabet
		a	b	c	d	e	f	
sequence *s*	**1**	1	1	1				g
	2				1	1	1	h
	3					1	1	h
	4				1	1		h
	5		1	1	1			g

Table 3.4 New codebook and alphabet obtained from the set of vectors that represent *s* in Table 3.3

Space						Alphabet
a	b	c	d	e	f	
1	1	1				g
			1	1	1	h

Figure 3.4 shows on the left the representation of sequence *s* under alphabet {g, h}, after applying the temporal grouping procedure once more, the final representation of sequence *s* under alphabet {k} appears on the right, which is simply *s*=k, a single point in

the last space time domain. At this point the construction of automaton *A* is complete; now, *A* should be able to accept sequences similar to *s* and reject those that don't resemble it.

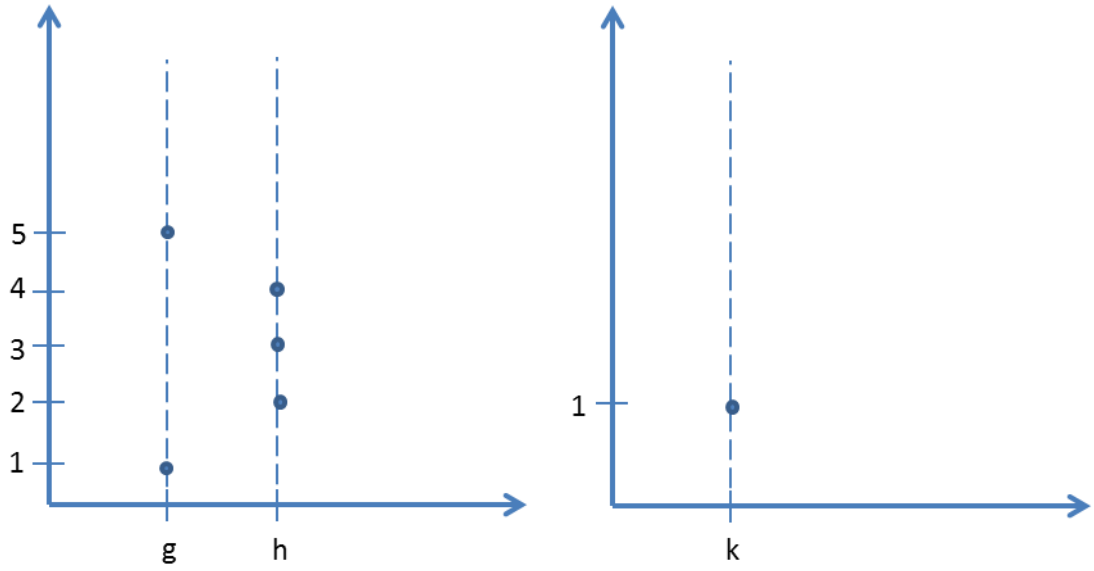

Figure 3.4 Representations of sequence *s* under alphabet {g, h} and after one more temporal grouping under alphabet {k}

3.2.3 Properties of the Automaton A

3.2.3.1 Semantic Representation

The data is promptly converted into a semantic representation using the mapping $h_{1,2}: \Lambda \rightarrow \{0,1\}^m$ that maps a high dimensional lattice into a high dimensional Boolean space where a point in the lattice corresponds to a bit in the Boolean vector. This conversion is costly in terms of storage, but necessary to achieve the property. Once the representation is

semantic, measuring similarity is straightforward. This property remains consistent across the hierarchy.

3.2.3.2 Autoassociativity

Autoassociativity is defined as the ability to associate a pattern with a part of itself. The model obtains this property from its semantic representation and by using the set of codebooks Γ_i, partial inputs have semantic similarity with already stored codewords, enabling the recovery of a complete pattern. Since this property remains consistent across the hierarchy, its influence immediately extends to space and time.

From a cognitive point of view, autoassociativity as a property of the brain might give the explanation to why analogies are so important for cognition [31]. A new concept, difficult at first, with the right analogy is immediately associated with an already mastered concept, as if the new concept is transformed into a partial input of the mastered concept and autoassociated to it.

3.2.3.3 Noise tolerance

The model obtains this property for the same reason that it obtains autoassociativity, the use of semantic representations and codebooks across the hierarchy. Semantic similarity and codewords enable the restoration of the pattern to a version that is useful to the model. Also, since this property is consistent across the hierarchy, its influence extends to time and space.

3.2.3.4 Semantic Invariance

Autoassociativity and noise tolerance are important steps toward invariant semantic representations. Missing or distorted parts in an input sequence, in general in lower levels of the hierarchy, can be completed or restored by autoassociativity and noise tolerance in higher levels of the hierarchy, avoiding interruption or failure of the recognition process.

An invariant semantic representation seems to be the characterization of a region of space time for a given input; after that, new inputs that fall within this region produce the same invariant representation. If the model is still learning, this region of space time is dynamically adjusting its shape to the more frequent inputs.

From a cognitive point of view, invariant semantic representations could define what a mental abstraction is, a characterized region of space time. Since all these representations have a homogenous nature, it is perfectly valid to feed abstractions, as representations, into other hierarchies which could be the foundation for high level thinking.

3.2.3.5 Input prediction

The model has one deterministic finite automaton for every level of the hierarchy; these automata are in charge of prediction duties. They learn their transition functions from the training data that later dictates what to expect. Using finite automata in such a natural way is only possible due to the semantic invariance property that enables the use of alphabets based on invariant representations.

Originally, the model rejects a sequence if any level in the hierarchy rejects the sequence. This constraint can be relaxed in lower levels of the hierarchy and maintained in higher levels, the model can be more noise tolerant with this modification. Additionally, in a well-trained model, a rejection of a sequence at a higher level of the hierarchy might be considered as an anomalous input or an outlier.

3.2.3.6 Subsequence contextualization

This aspect is not considered in the theoretical model, but can be achieved in the implementation considering the technique of splitting states [32]. The problem of subsequence contextualization refers to the following situation:

Given sequences *s*=abcd and *r*=ebcf the model should be able to predict d after c, only if a was at the beginning of the contextualized sequence; similarly predict f after c, only if e was at the beginning of the contextualized sequence. Otherwise, unacceptable sequences as {ebcd, abcf} might be accepted.

Splitting states implies that every time a context is detected, new states are created for the subsequence in the new context, with the condition that these states should respond to the same corresponding input in the original subsequence. Transitions between sequence elements are learned independently for every context. This is equivalent to transforming original sequences *s* and *r* into *s*=ab_1c_1d and *r*=eb_2c_2f. All b_i respond to b but they only activate depending on the previous sequence element, once a context is activated, it is clear

to the model what sequence element should be next. The strategy to detect a new context is: if two or more states predict the same state `a` then split `a`.

3.3 Model Implementation

Two implementations were developed; the first implementation uses sparse distributed representations and has similarities to the HTM implementation [24]. The second implementation strictly uses a vector quantization approach based on thresholds. An important goal of these implementations is to construct an instance of automaton A almost exclusively by *learning* from data of the training set E. The first implementation makes two important considerations:

- The input data is already in Boolean format as a representation in $\{0,1\}^m$. Mappings $h_{1,1}: \Omega \to \Lambda$ and $h_{1,2}: \Lambda \to \{0,1\}^m$ can be done independently and specifically for every experiment. m is expected to be in the order of thousands.
- Mappings $h_{1,4}: \Gamma_1 \to \Sigma_1$ and $h_{i,4}: \Gamma_i \to \Sigma_{i+1}$ are not performed. Codebooks Γ_i are implemented via sparse representations over the Boolean space with a sparsity factor of 2%. Codebooks cardinalities should be decreasing, $|\Gamma_i| > |\Gamma_{i+1}|$. Corresponding implementations of automata M_i use codebooks Γ_i as if they were alphabets.

The neocortex favors sparse distributed representations due to two important reasons:

- To achieve semantic representations. Sparse representations have the ability to uncover semantic information due to a simple but important property of very high dimensional data, at least observable in many image applications: instances belonging to the same class exhibit a degenerate structure, meaning that they lie on or near low-dimensional subspaces or submanifolds [33].
- Using strictly an alphabet would be equivalent to using a local code, in this case, a single neuron. A damaged neuron in a local code could completely erase a portion of a memory. Instead, using a sparse representation can be seen as a measure for fault-tolerance; a damaged neuron in a sparse code only produces degradation in a memory.

The second implementation strictly uses a vector quantization approach based on thresholds. These two implementations show the generality of the model, allowing the use of different methods to compute codebooks. Essentially, any method that can group vectors and obtains representatives of those groups can be used.

3.3.1 *First Implementation*

The hierarchy and its main components are represented in Figure 3.5. The hierarchy is made by a set of layers; every layer has a set of columns, where every column has a specific feed forward input, taken as a random subset of the whole input. A column has a group of cells that respond to the input of the column. A cell has a set of segments of dendrites for prediction purposes; a segment remembers what columns were active just before the cell became active. The sparse code is achieved by selecting a small set of columns that best match the current input, the rest of the columns are inhibited. The set of winner columns is rewarded to improve its response to the particular input.

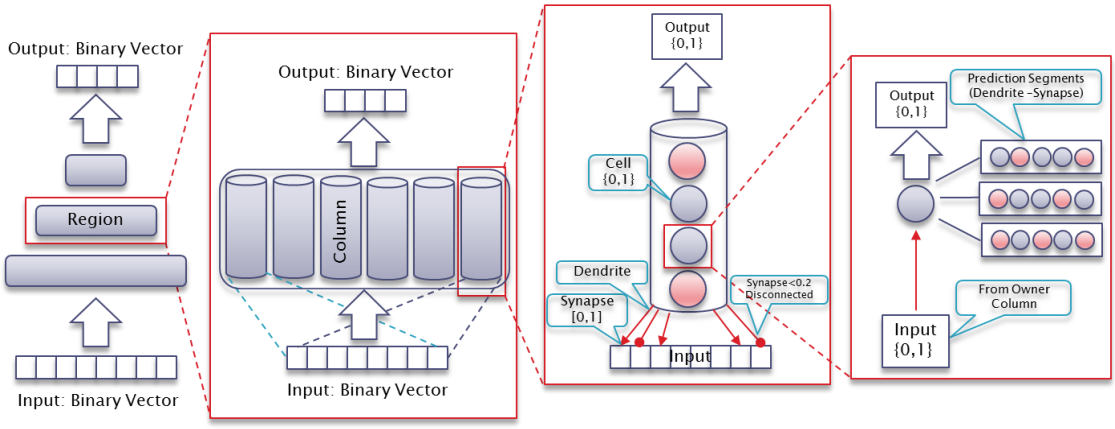

Figure 3.5 First implementation design. The hierarchy and its main components are schematized.

The main classes used in this implementation are:

- *Hierarchy* = (*Input, Regions, Output*).
    - *Input* reads a sequence of Boolean vectors representing the initial entry for the model. Every time a sequence element is received, a global time step is incremented, triggering a sequential attempt by *Hierarchy.Regions* to process the sequence element, some regions may have to wait to receive enough sequence elements before computing successfully an output.
    - *Regions* is a list of instances of class *Region*.
    - *Output* returns the label for a given entry sequence once it is completely read.
- *Region* = (*Input, Columns, Output*).
    - *Input* reads a sequence of Boolean vectors from *Hierarchy.Input* or from a previous *Region.Output*.
    - *Columns* is a list of instances of class *Column*.
    - *Output* returns a sparse representation as a Boolean vector to feed the next *Region.Input* or *Hierarchy.Output*. *Region* uses *Column.Output* to build a sparse representation, a small subset of Columns, determined by the sparsity factor and under the strategy "winners take all". *Region.Output* is computed every time a sequence element is read.
- *Column* = (*Input, Cells, Dendrites, Synapses, Output*)

- *Input* receives a sequence element, as a Boolean vector, from *Region.Input*.
- *Cells* is a list of instances of class *Cell*. This attribute enables state splitting, to handle different context in which a *Column* might be part of.
- *Dendrites* is a Boolean vector with the same dimensions as *Column.Input*. This attribute represents the potential connections (0 not connected, 1 potential connection) that a *Column* can establish to the input.
- *Synapses* is a vector with the same dimensions as *Column.Input*. This attribute, in conjunction with *Column.Dendrites*, states if a potential connection is, at the moment, connected or not, to corresponding positions in the input. The components of this vector take values in the real interval [0, 1], are considered connected when values are above a threshold (0.2 in experiments), and can be modified by learning.
- *Output* is a non-negative integer value that measures matching between the connected part of *Column.Dendrites* and *Column.Input*. This attribute is computed as:

$$Output = Input \cdot (Dendrites \wedge connected(Synapses))$$

where $(\cdot)$ is the usual dot product between vectors. If *Column.Output* is strong enough to be part of winner columns, *Column* checks its attribute *Cells* and

activates only those in predictive state; if no *cell* is in predictive state, all *cells* in *Column* are put in active state.

- *Cell* = (*Segments, ActiveState, PredictiveState, LearnState*)
    - *Segments* is a list of instances of class *Segment*. A *Segment* links a sparse representation to a cell that might be activated in the next time step due to the next sequence element, works like a prediction mechanism.
    - *ActiveState* indicates if the cell is active or not at the end of current time step.
    - *PredictiveState* indicates if the cell is predicting its activation or not at the beginning of current time step. Computation occurs in the end of previous time step and it is enough that one *Segment* in *Cell.Segments* is active to set this attribute on.
    - *LearnState* indicates if the cell is chosen for learning or not at the end of current time step.
- *Segment* = (*Dendrites*, *Synapses*)
    - *Dendrites* is a list indicating a set of cells whose activation usually precedes the activation of the owner *cell*.
    - *Synapses* is a list with the same dimensions as *Cell.Dendrites*. The elements of this list take values in the real interval [0, 1], are considered connected when

values are above a threshold (0.2 in experiments), and can be modified by learning.

A *Segment* is said to be active if the number of active cells for connected synapses in *Segment.Dendrites* is above a threshold.

The most important algorithms are offered in pseudo-code:

```
Region.ComputeSparseCode
Input:
      s: A sequence element
      k: The number of sparse columns to be retrieved
Output:
      c: A sparse code
1. For every Column in Region
   1.1.      Compute Column.Output for s
2. Select k winner Columns, the ones with the highest Column.Output
3. For every winner Column
   3.1.     Reward Column.Synapses that match s
   3.2.     Punish Column.Synapses that don't match s
4. Return the list of winner Columns as c
```

```
Region.ComputePrediction
Input:
      c: A sparse code
Output:
      p: Prediction as a sparse code
1. For every Column in c
   1.1.     If a cell in Column is in predictive state
      1.1.1.      Then activate cell and mark cell for learning
      1.1.2.      Else activate Column
2. If a Column was activated
   2.1.     Then select the cell with a segment that best matches the sparse
       code c
   2.2.     If no cell matches the sparse code c
      2.2.1.      Then choose a new cell and create a segment in it that
          matches the sparse code c
3. If a Cell was activated and was predicted
   3.1.     Then Reward the segment that best matches the sparse code c
4. If a Cell was predicted and not activated
   4.1.     Then Punish the segment that caused the prediction
5. Given current active cells compute upcoming predictions
```

3.3.2 Second Implementation

The hierarchy and its main components are represented in Figure 3.6. The hierarchy is represented by a set of layers; every layer has a codebook and a DFA associated with it. The DFA uses the codebook as an alphabet.

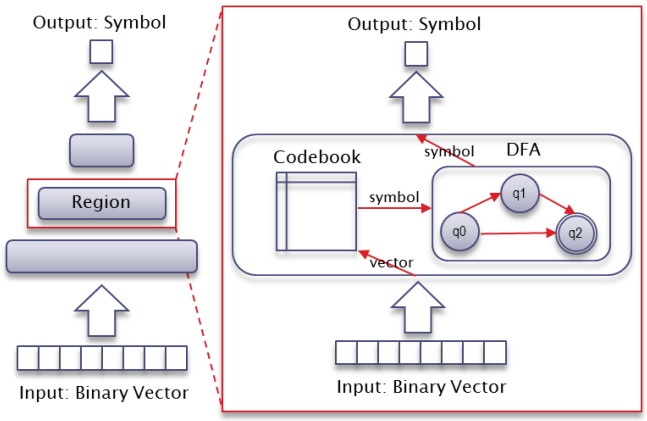

Figure 3.6 Second implementation design. The hierarchy and its main components are schematized.

This implementation is more compact than the previous one. The classes *Hierarchy* and *Region* remain; however, the class *Region*, instead of class *Columns*, has the classes *Codebook* and *DFA*. *Codebook* tries to match an input to and already stored codeword within a threshold; if there is no codeword that matches the input, a new codeword using the input is created. *DFA* uses, as a symbol of an alphabet, a unique identifier for every codeword; it creates the necessary states to learn correctly the transitions between symbols. The resulting

transition function is used later for input prediction; predicted symbols can have its threshold tolerance increased.

4 EXPERIMENTAL EVALUATION

4.1 Methodology

The goal of the experiments is to show and test specific aspects of the theoretical model under implementations that follow the specifications listed in Section 3.3. These aspects are:

- Construction of sparse distributed representations or codebooks using learning.
- Importance of order in training data during learning.
- Pattern reconstruction.
- Spatial and temporal autoassociativity.
- Spatial and temporal noise tolerance.
- Input prediction.
- Subsequence contextualization.
- Successful recognition across the hierarchy.

All these aspects are covered across six experiments. The experimental report format includes:

- Description.
- Properties considered.
- Data and parameters used.

- Results.
- Discussion.

Experiments I, II, III, IV and V are accomplished using the first implementation; these experiments have a qualitative character, showing the correct operation of the model. Experiments accomplished with the first implementation have these specifications in common: Every column has 6 cells. Dendrites of a column randomly cover 75% of the input, 60% of these dendrites have connected synapses. Every cell has 12 segments. Sparsity factor is 2% for all layers. The input is always a 50×50 binary image per time step.

Experiment VI is accomplished using the second implementation; this experiment has a quantitative character, showing the performance of the model as a classifier for a real dataset, the dataset was taken from the Carnegie-Mellon Motion Capture Database [34].

4.2 Experiment I

4.2.1 Description

Given a simple sequence *S* of 2D binary images as a training data set, train the model and show the steps that create sparse distributed representations. The binary images represent single uppercase letters. Test the model using different versions of sequence *R*, where individual patterns have different levels of salt and pepper noise or have been cropped. Show pattern reconstruction using the learned sparse code.

4.2.2 Properties considered

Construction of sparse distributed representations using learning, importance of order in training data during learning, pattern reconstruction, spatial autoassociativity and spatial noise tolerance.

4.2.3 Data and parameters used

- Model: 1 layer (L1). Layer dimensions: L1=32×32 columns. Sparsity factor is returning 21 columns for every pattern. Learning is done with 4 iterations.
- Training sequence: S=`AFM`
- Testing sequence: R=`AF`
- Noise levels: A random set N of values switched in the testing sequence, where $|N|$ is 10%, 30%, 50% and 70% of nonzero values in the original binary image.
- Cropping: 50% of nonzero values, lower or right halves in the testing sequence.
- Pattern reconstruction: Majority vote. A pixel is accepted if more than half of the sparse code accepts it.

4.2.4 Results

After initialization, Figure 4.1 (a) shows the potential input coverage; in average 75% of columns has a dendrite to a specific input pixel. (b) shows the effective input coverage; in average 45% of columns has a dendrite with a connected synapse to a specific input pixel.

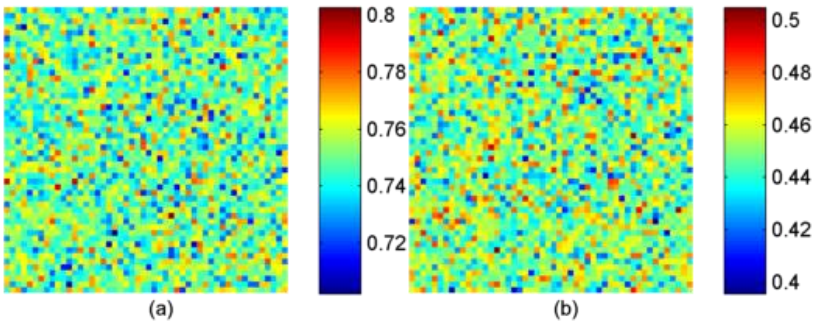

Figure 4.1 (a) Potential input coverage. (b) Effective input coverage.

Before any learning, Figure 4.2 shows the dendrites over the input of one specific column. (a) shows the whole set of dendrites. (b) shows the initial value of synapses in the range of [0.18, 0.23]. (c) shows connected synapses, with values equal or greater than 0.2.

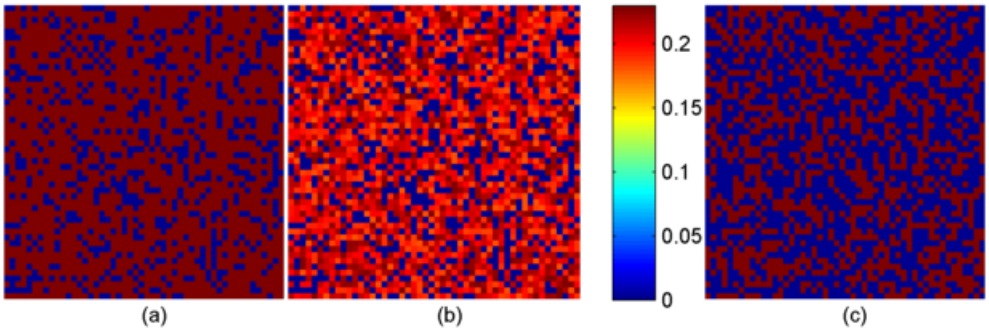

Figure 4.2 Dendrites of a column after initialization. (a) Dendrites. (b) Values of synapses. (c) Connected synapses.

Figure 4.3 shows the changes over the synapses of one winning column after one iteration. Given the input pattern, hits and misses are rewarded and penalized respectively by changing the strength of synapses.

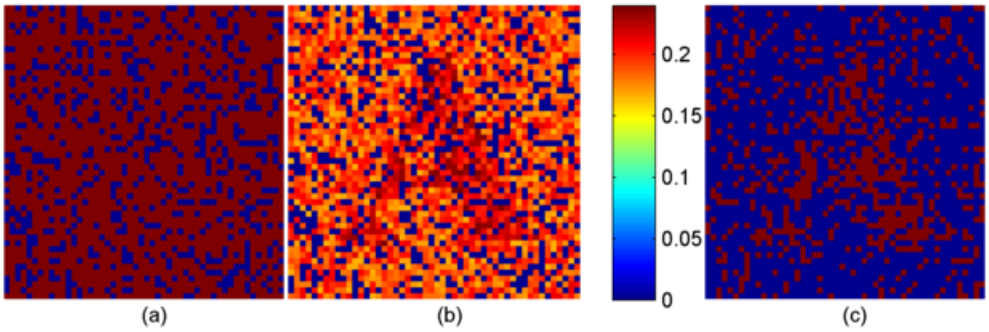

Figure 4.3 Dendrites of a column after one iteration. (a) Dendrites. (b) Values of synapses. (c) Connected synapses.

After two iterations, Figure 4.4 shows the changes over the synapses of one winning column. The remaining connected synapses finally match the input in the best possible way.

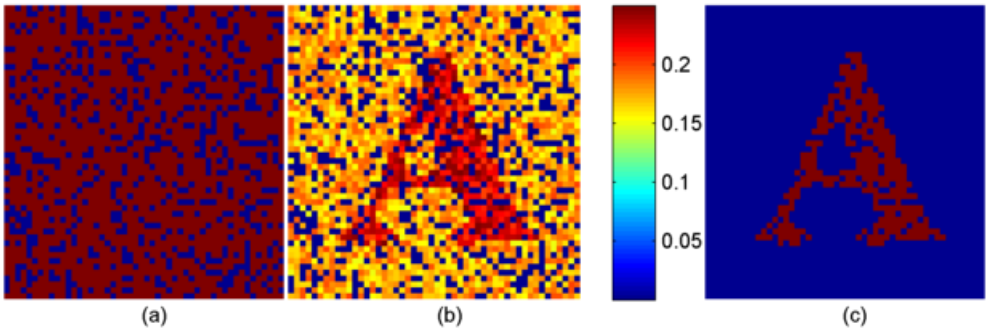

Figure 4.4 Dendrites of a column after two iterations. (a) Dendrites. (b) Values of synapses. (c) Connected synapses.

Once the training is finished, the sparse code for pattern “A”, represented by a set of 21 winning columns, can be used to reconstruct the learned pattern under a majority vote criterion. Figure 4.5 shows the reconstruction process. (a) shows the voting according the

sparse code, a pixel has a voting in the range 0 to 21. (b) shows the binary reconstruction after majority vote, a pixel needs 11 or more votes to be accepted. (c) shows the reconstruction error, for pattern "A" the error is just one pixel.

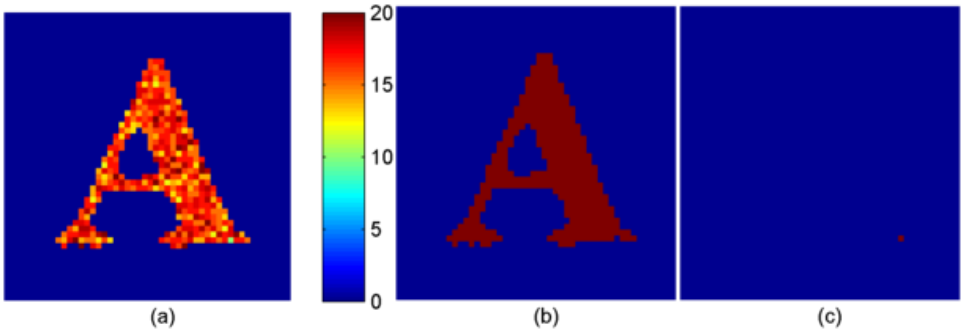

Figure 4.5 Pattern reconstruction. (a) Voting according the sparse code. (b) Binary reconstruction after majority vote. (c) Reconstruction error.

To test tolerance to noise, different levels of random noise are introduced in the input. Noise is in terms of percentage of nonzero values in the pattern. Pattern "A" has 377 nonzero values; a 10% spatial noise means that 38 pixels are switched in the pattern. Figure 4.6 and Figure 4.7 show the testing of an input with spatial noise of 10% and 30% respectively. In both cases, (a) shows the input with noise. (b) shows the voting according the sparse code obtained for the input. (c) shows the binary reconstruction after majority vote. The reconstruction error is 1 pixel, the same as in the original learned sparse code.

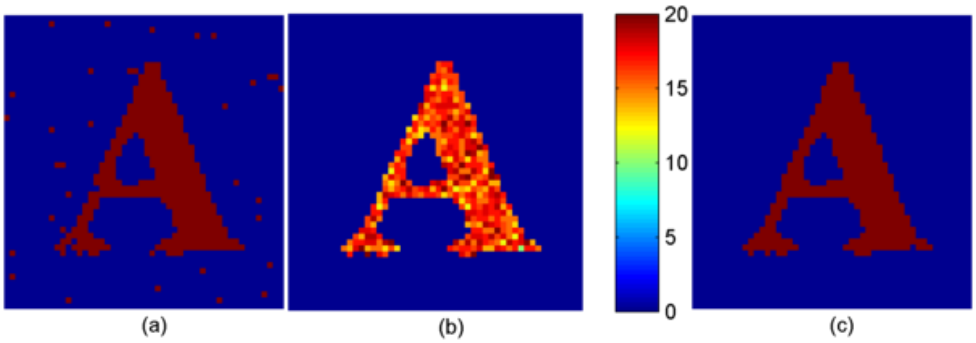

Figure 4.6 Testing spatial noise. (a) Input with 10% noise. (b) Voting according the resultant sparse code. (c) Binary reconstruction after majority vote.

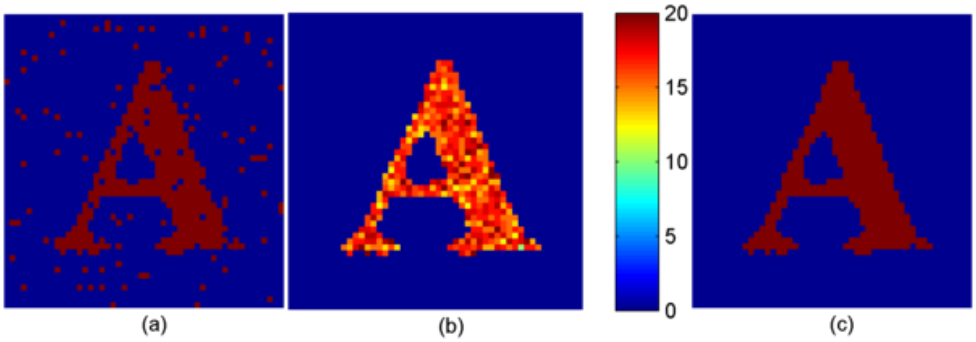

Figure 4.7 Testing spatial noise. (a) Input with 30% noise. (b) Voting according the resultant sparse code. (c) Binary reconstruction after majority vote.

At 50% level of noise, Figure 4.8 (b) shows how the sparse code obtained for the input pattern includes untrained columns that are actually matching the noise, yet (c) shows that majority vote leads to a successful reconstruction of the pattern. Reconstruction error is 1 pixel.

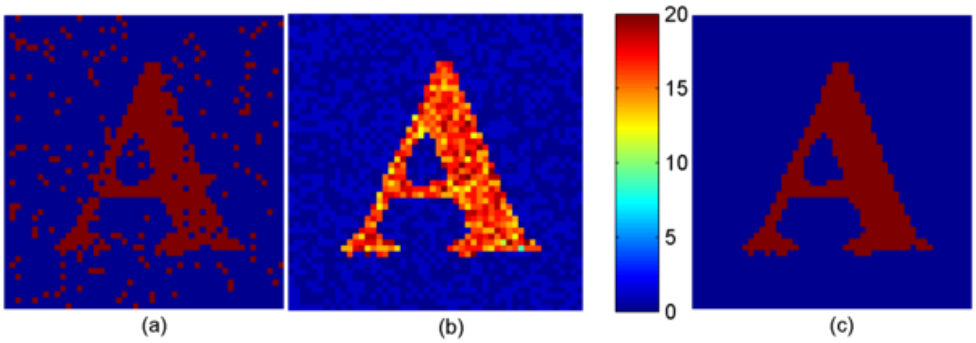

Figure 4.8 Testing spatial noise. (a) Input with 50% noise. (b) Voting according the resultant sparse code. (c) Binary reconstruction after majority vote.

At 70% level of noise, Figure 4.9 (b) shows that the sparse code obtained for the input pattern no longer resembles the original pattern "A", most columns in the sparse code are untrained columns producing a random pattern in the voting. (c) shows that majority vote produce an unrecognizable pattern "A". Reconstruction error is 715 pixels.

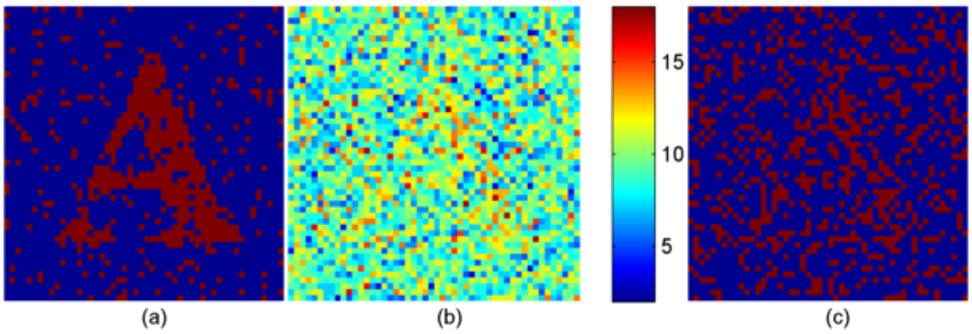

Figure 4.9 Testing spatial noise. (a) Input with 70% noise. (b) Voting according the resultant sparse code. (c) Binary reconstruction after majority vote.

To test autoassociativity, three incomplete instances of pattern “F” are submitted as input. Figures Figure 4.10, Figure 4.11, and Figure 4.12 show how the model is capable of recovering the original learned pattern from an incomplete version of itself. The reconstruction error in all cases is 1 pixel, the same as in the original learned sparse code.

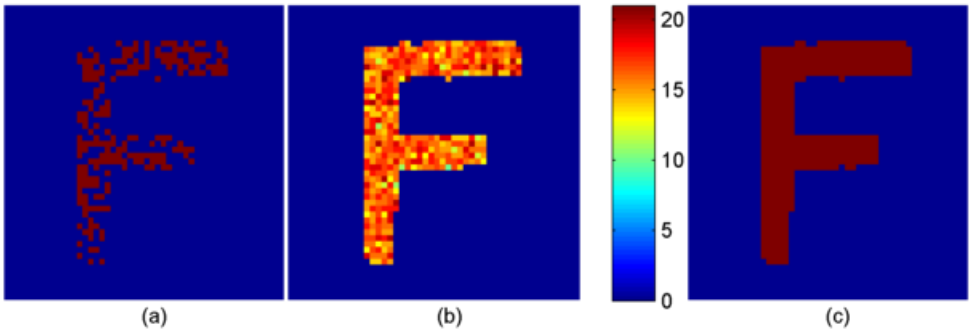

Figure 4.10 Testing autoassociativity. (a) Input with 50% pixels cropped. (b) Voting according the resultant sparse code. (c) Binary reconstruction after majority vote.

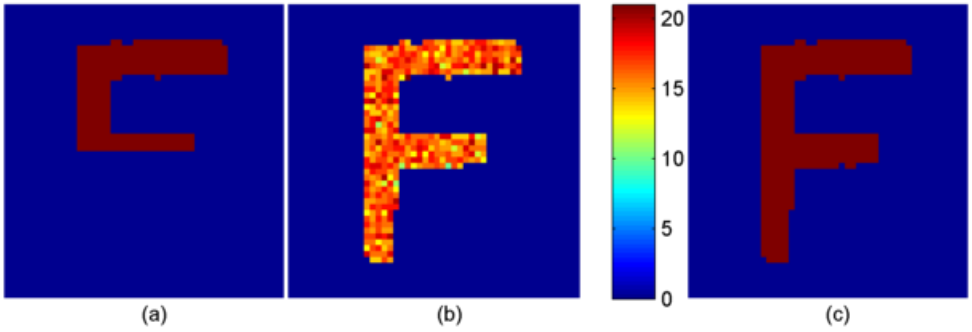

Figure 4.11 Testing autoassociativity. (a) Input with lower half cropped. (b) Voting according the resultant sparse code. (c) Binary reconstruction after majority vote.

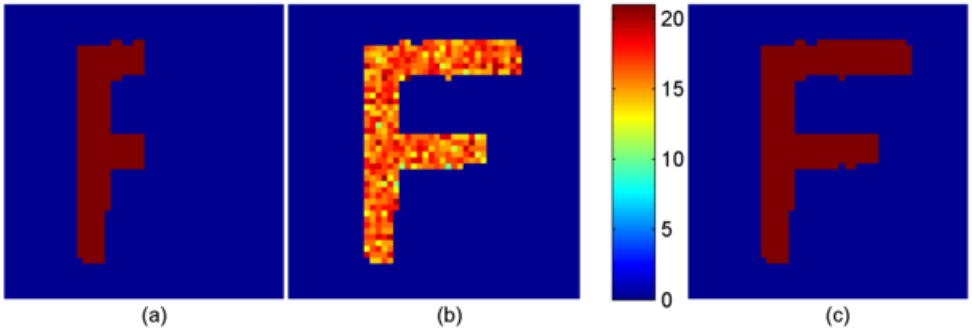

Figure 4.12 Testing autoassociativity. (a) Input with right half cropped. (b) Voting according the resultant sparse code. (c) Binary reconstruction after majority vote.

4.2.5 Discussion

This experiment demonstrates two desirable properties of the pattern recognition model, spatial noise tolerance and spatial autoassociativity; it also shows the importance of noise free training data and well characterized patterns. In presence of abundant training samples, learning can automatically discard spatial random noise since it is not stable across different samples.

4.3 Experiment II

4.3.1 Description

Given sequences *S* and *R* of 2D binary images as a training data set, train the model and show that both sequences are consistently recognized across the hierarchy. These sequences have common subsequences that the model should be able to handle in proper context making the right predictions.

4.3.2 Properties considered

Subsequence contextualization and successful recognition across the hierarchy.

4.3.3 Data and parameters used

- Model: 3 layers (L1, L2, L3). Layer dimensions: L1=32×32 columns, L2=16×16 columns and L3=10×10 columns. Temporal parameters are 5, 4 and 1 for layers L1, L2 and L3 respectively. Learning is done with 2 iterations.
- Training sequences:
    - *S*=`ABGMLNRST_VXYZ_HMRQK`
    - *R*=`EBGWLNRST_MXYH_HMQRK`
- Testing sequences: *S* and *R*
- Common subsequences {`BG`,`LNRST_`,`XY`,`_HM`,`K`}

4.3.4 Results

The sparse codes, instead of binary images, are used to present the results. Sparse codes for predictions in L1 show consistent single pattern prediction in all cases. Consistent in the sense that last elements in common subsequences, patterns {G,_,Y,M,K}, are represented by the same sparse code (columns) in sequences *S* and *R*. Single pattern prediction implies that the model is correctly identifying the context when reading a common subsequence, thus using different learning cells inside the columns of the corresponding sparse code, allowing a unique prediction after the end of the common subsequence. Tables Table 4.1 and

Table 4.2 show also consistent single pattern predictions in L2, at this level there are two common subsequences {NRST_, HMRQK}. L3 recognizes correctly sequences *S* and *R* as in the training phase.

Table 4.1 Sparse Code in L2 and L3 for sequence *S*

Time Step	L2 Activations	L2 Predictions	L3 Activations
5	[59,89,144,149,150,162]	[21,41,59,131,154,216]	
10	[21,41,59,131,154,216]	[15,27,53,85,210,219]	
15	[15,27,53,85,210,219]	[2,5,9,36,48,117]	

20	[2,5,9,36,48,117]		[19,71]

Table 4.2 Sparse Code in L2 and L3 for sequence *R*

Time Step	L2 Activations	L2 Predictions	L3 Activations
5	[13,62,135,141,228,251]	[21,41,59,131,154,216]	
10	[21,41,59,131,154,216]	[33,39,61,139,200,229]	
15	[33,39,61,139,200,229]	[2,5,9,36,48,117]	
20	[2,5,9,36,48,117]		[9,49]

4.3.5 Discussion

Recognizing proper context is very important for the model to make accurate predictions of what will happen next, without context one pattern can trigger several predictions with many of them unacceptable for the current input.

4.4 Experiment III

4.4.1 Description

Given sequences *S* and *R* of 2D binary images as a training data set, train the model and use testing data set {S1,S2,R1,R2} to evaluate recognition across the hierarchy. The testing data set introduces temporal perturbations as missing elements, extra elements, changes in order and different values.

4.4.2 Properties considered

Temporal autoassociativity, temporal noise tolerance and successful recognition across the hierarchy.

4.4.3 Data and parameters used

- Model: 3 layers (L1, L2, L3). Layer dimensions: L1=32×32 columns, L2=16×16 columns and L3=12×12 columns. Temporal parameters are 10, 5 and 1 for layers L1, L2 and L3 respectively. Learning is done with 4 iterations.
- Training sequences:
  - *S*=ABCDEFGHIJKLMNOPQRSTUVWXYZ_ABCDHIJKLMNOPQRSTUVWXYZ
  - *R*=GHTSRPQNOPQRSTUABCDEFABCD_HIJKLMNOPZXYWVUABCSOPQGE
- Testing sequences:
  - *S1*=ABCEFGHIJKLMNOQRSTUVWXYZ_ABCDHI_JKLMNO_PQRST_UVXYZ
  - *S2*=ABDCEFGHIJKAMNOPQTRSUVAXYZ_ABCDHLAJKMNOPQURATVWXYZ
  - *R1*=GHTSRPNOLQR_STUABDEFABCD_HIJKBMNOPZBYWVUABCSOBQGE_
  - *R2*=GHTSRPCQNPOQRSTUABCDFEABC_HIJKLXMNOPZXVYWUABCSPOQG

4.4.4 Results

The sparse codes, instead of binary images, are used to present these results. Since the input for L1 represents 1 time step, temporal analysis is meaningful only for higher levels. Due to particular temporal parameters for this experiment the input to L2 represents

sequences of 10 time steps and the input to L3 represents sequences of 50 time steps mapped to just 5 L2-patterns. Tables Table 4.3 and Table 4.4 compare sparse codes of training data and testing data. Some L2-patterns are not identical, still applying majority vote in L3 it is acceptable to say that *R*, *R1* and *R2* are in the same class.

Table 4.3 *S*, *S1* and *S2* sparse codes

Time Step	Seq	L2	L3
10	*S*	[9,41,105,118,139,154]	
	S1	[9,41,105,118,139,154]	
	S2	[9,41,105,118,139,154]	
20	*S*	[113,182,188,193,202,222]	
	S1	[43,57,59,112,138,190]	
	S2	[92,113,188,193,202,222]	
30	*S*	[17,28,64,69,167,210]	
	S1	[17,28,64,69,167,210]	
	S2	[17,28,64,69,167,210]	
40	*S*	[36,61,85,87,174,226]	
	S1	[36,61,85,87,174,226]	
	S2	[36,61,85,87,174,226]	
50	*S*	[20,62,128,152,214,228]	[89,90,144]
	S1	[20,62,128,152,214,228]	[89,90,144]
	S2	[20,62,128,152,214,228]	[89,90,144]

Table 4.4 *R, R1* and *R2* sparse codes

Time Step	Seq	L2	L3
10	*R*	[113,182,188,193,202,222]	
	R1	[113,188,190,193,202,222]	
	R2	[113,188,202,222,227,255]	
20	*R*	[31,53,89,133,192,234]	
	R1	[31,57,79,93,117,178]	
	R2	[31,34,89,112,133,138]	
30	*R*	[9,41,105,118,139,154]	
	R1	[9,41,105,118,139,154]	
	R2	[9,33,41,105,118,139]	
40	*R*	[35,146,159,207,219,233]	
	R1	[35,146,159,207,219,233]	
	R2	[35,146,159,207,219,233]	
50	*R*	[34,53,89,133,192,234]	[2,72,112]
	R1	[34,53,89,133,192,218]	[72,85,112]
	R2	[3,34,111,112,124,133]	[72,112,127]

4.4.5 Discussion

In some cases the sequences are considerably distorted, but due to sparse coding over a semantic representation, the similarity between patterns degrades gradually (in time and space), however higher levels can still use the representation.

4.5 Experiment IV

4.5.1 Description

Given sequences *S* and *R* of 2D binary images as a training data set, train the model and use testing data set {S1, R1} to show how the model can focus on temporal structure. All sequences involve small spatial patterns. Testing sequences introduce temporal perturbations compared with corresponding training sequences.

4.5.2 Properties considered

Spatial and temporal autoassociativity, temporal noise tolerance.

4.5.3 Data and parameters used

- Model: 3 layers (L1, L2, L3). Layer dimensions: L1=32×32 columns, L2=16×16 columns and L3=12×12 columns. Temporal parameters are 10, 5 and 1 for layers L1, L2 and L3 respectively. Learning is done with 2 iterations.
- Training sequences: *S* and *R*. These sequences contain 32 time steps, they represent a path followed by a small square. Figure 4.13 shows parts of these sequences.

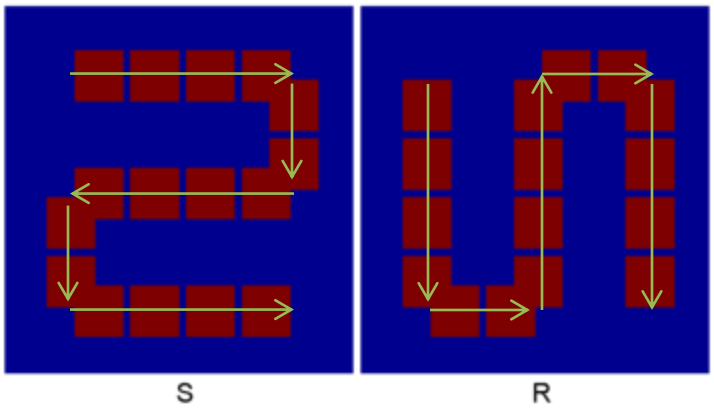

Figure 4.13 Partial illustration of training sequences *S* and *R*

- Testing sequences: *S1* and *R1*. These sequences contain 32 time steps, they represent a perturbed path followed by a small circle. Figure 4.14 shows these sequences

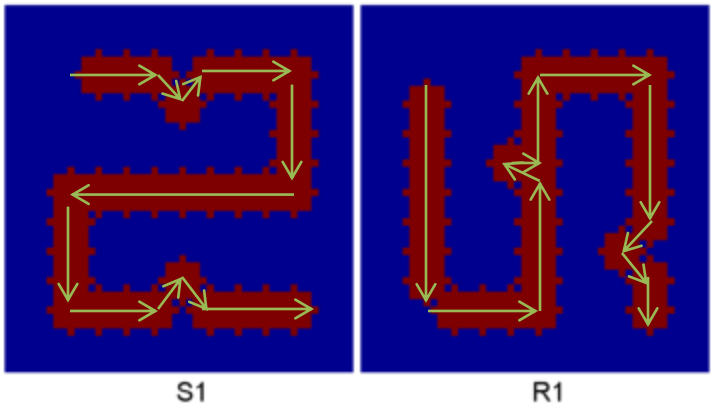

Figure 4.14 Illustration of testing sequences *S1* and *R1*

4.5.4 Results

The sparse codes for levels L2 and L3 are used to present these results. Due to particular temporal parameters for this experiment the input to L2 represents sequences of 8

time steps and the input to L3 represents sequences of 32 time steps mapped to just 4 L2-patterns. Tables Table 4.5 and Table 4.6 compare sparse codes. *S* and *S1* are recognized as the same sequence; also *R* and *R1* are recognized as the same sequence.

Table 4.5 *S* and *S1* sparse codes

Time Step	Seq	L2	L3
8	*S*	[6,13,55,71,118,230]	
	S1	[13,55,71,118,230,254]	
16	*S*	[103,111,140,153,243,248]	
	S1	[103,111,140,153,243,248]	
24	*S*	[96,112,163,206,218,222]	
	S1	[96,112,163,206,218,222]	
32	*S*	[30,48,117,135,139,158]	[24,57,135]
	S1	[30,48,117,135,139,158]	[24,57,135]

Table 4.6 *R* and *R1* sparse codes

Time Step	Seq	L2	L3
8	*R*	[57,75,96,131,202,251]	
	R1	[2,57,75,131,202,251]	
16	*R*	[29,65,132,180,191,241]	
	R1	[29,65,132,180,191,241]	
24	*R*	[6,26,34,142,150,182]	
	R1	[6,26,34,142,150,182]	
32	*R*	[25,32,136,141,177,210]	[85,133,144]
	R1	[25,32,136,141,177,210]	[85,133,144]

4.5.5 Discussion

Using small spatial elements allows focus on temporal structure regardless of spatial structure; spatial autoassociativity recovers the original learned pattern. This setting is useful for learning paths across the 2D domain.

4.6 Experiment V – Short Video Sequence Recognition

4.6.1 Description

Given the training data set {S, R, T}, the model is tested using the data set {S1, R1, T1} to evaluate recognition across the hierarchy. The data represent short video sequences (originally in *.gif* format); all sequences have 18 color frames. The frame size is 200×200 pixels. An RGB pixel is converted to a Boolean representation using the mapping intervals [0, 0.2, 0.4, 0.6, 0.8, 1] in every channel. The result gives a 15-component vector, with 3 active bits. The values used for reconstruction in corresponding intervals are (0, 0.25, 0.5, 0.75, 1).

4.6.2 Properties considered

Spatial and temporal autoassociativity, spatial and temporal noise tolerance.

4.6.3 Data and parameters used

- Model: L1=32×32, L2=16×16 and L3=12×12 columns. Temporal parameters are 6, 3 and 1 for layers L1, L2 and L3 respectively. Learning is done with 4 iterations.

- Training sequences (Figure 4.15):

  S: weight lifting. R: jumping. T: running.

Figure 4.15 Training sequences. Weight lifting, jumping, running.

- Testing sequences (Figure 4.16):

  S1: weight lifting plus a red text. R1: jumping plus snow effect. T1: running plus rain.

Figure 4.16 Testing sequences. Different types of noise introduced.

4.6.4 Results

The whole testing data set is recognized successfully.

4.6.5 Discussion

Despite the different types of noise introduced in every frame of the sequence, the model is able to recognize the sequence. The reconstruction of the sequence, using the set of

reconstruction values, leads to a noise free sequence, with a color pretty close to the original sequence.

4.7 Experiment VI – Motion Capture Data Classification

4.7.1 Description

Given a training set E and a testing set T, the model is trained using E to perform a classification over the testing set T. There are 7 classes, training set E has one sequence per class, and testing set T has 42 sequences. Classes have an uneven distribution of samples. The dataset comes from the MoCap Database [34].

4.7.2 Data and parameters used

The data is in *skeleton movement* format (*.amc* filetype). Every time step in a sequence returns 62 real-values as a vector. All sequences have different length. The data capture human activities (Figure 4.17) such as walking (19 samples), running (11), jumping (8), walking in an uneven terrain (2), basketball forward dribbling (3), basketball backward dribbling (2), and soccer ball kicking (4). This data produces a real matrix with 62 columns and as many rows as time steps are in the sequence. The preprocessing stage is quite simple, for every column the maximum and minimum values are obtained; these values determine a set of intervals. Every interval is divided into 10 equal portions; every portion is mapped to a Boolean vector with 10 bits. For a given real value, the bit representing the corresponding portion is set to 1 and the rest to 0. After preprocessing, every time step in a sequence has a

representation as a Boolean vector with 620 components, where just 10 bits can be set to 1 at any given time step.

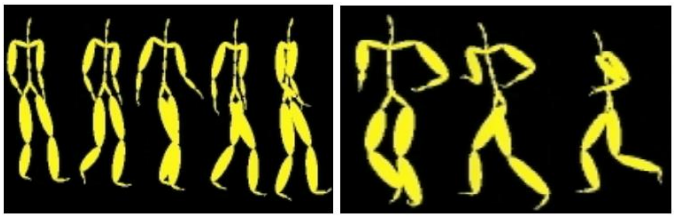

Figure 4.17 Two skeleton sequences. Walking and running.

The model contains 3 levels: L1, L2, and L3. Every level has the same 75% similarity threshold. Temporal parameters are 50, 12, and 1. One iteration is sufficient to train the model.

4.7.3 Results

The classifier achieved, in average, an accuracy of 92.4%. The validation used was *repeated random sub-sampling* with 10 trials. All selected random training sets contained one sequence per class, the rest of the sequences were used in the testing sets. Table 4.7 Confusion matrix for the testing set shows the average confusion matrix for the testing set in 10 trials. Classes are: 1. Walking. 2. Running. 3. Jumping. 4. Walking in an uneven terrain. 5. Basketball forward dribbling. 6. Basketball backward dribbling. 7. Soccer ball kicking.

Table 4.7 Confusion matrix for the testing set

	1	2	3	4	5	6	7
1	0.88	0.04	0.14	0.20	0.30	0.00	0.00
2	0.00	1.00	0.00	0.00	0.00	0.00	0.00
3	0.01	0.00	0.97	0.00	0.00	0.00	0.00
4	0.00	0.00	0.04	0.70	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	1.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.15	0.70	0.00
7	0.00	0.00	0.03	0.00	0.00	0.00	0.93

The accuracy results are good, considering the fact that just one sample per class was used in the training set and all the sequences have different lengths. The model seems to be a strong classifier for this kind of data and it can compete with methods such as [35] based on Hidden Markov Models which had an accuracy of 95% on other motion capture data set.

4.8 Running time

The main factor that determines running time is the length of the sequence. The first implementation (Experiments I to V) has an average running time of 0.2 seconds per time

step. The amount of cells involved in learning can also be considered; but, since this number is related to the sparsity factor, its impact is small compared to the whole number of cells in a level. The second implementation (Experiments VI) shows an important improvement in running time, a time step takes in average 0.005 seconds. Also, the second implementation is more efficient compared to the first one in terms of storage. The first implementation has to preallocate a big amount of memory in order to initialize its columns and cells; on the other hand, codebooks use an on-demand scheme to allocate more memory in real-time. All tests are run on a computer with an Intel Core i5 1.8 GHz CPU, 6GB RAM.

5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This thesis presented a generic model for pattern recognition, mainly based on the known operation of the neocortex. The theoretical model also establishes a more general framework for treatment of space time data through a dimensionality reduction process.

The model exhibits desirable properties for a pattern recognition system, such as spatial and temporal autoassociativity, spatial and temporal noise tolerance, recognition under sequence contextualization, and input prediction.

The last experiment shows the model as a strong classifier in a supervised classification case; achieving an accuracy of 92.4% in average. For this experiment, the training set had only one sample per class.

The way the model characterizes a region of space time for a given input, defines something that might be called an invariant semantic representation. At the same time, these invariant semantic representations could shed light on what a mental abstraction is. Autoassociativity might give an explanation to the usefulness of analogies as a tool for cognition. Abstractions and analogies are concepts that definitely find an important place within the effort of the brain to build a model, a representation, of the real world in order to survive.

From the two implementations, the first one has a running time of 0.2 seconds per time step. The second implementation shows a great improvement over the first one; having a promising running time of 0.005 seconds per time step, low on-demand requirements for storage, and the capacity to be trained in just a single-scan.

5.2 Future Work

The following considerations are presented as future work:

- Bidirectional processing of information across the hierarchy to improve predictions. Current model and implementation only consider unidirectional (bottom-up) processing of information.
- Variable temporal grouping of sequences for a more effective dimensionality reduction of data. Current implementation works with fixed temporal parameters, forcing the length of sequences to be a multiple of temporal parameters.
- Extend the model to include active representations. Current model only considers passive representations in the sense that it only receives information to recognize. Active representations include actions to perform when the memory is recovered. Brains store active representations that communicate with the motor cortex to trigger movement.

- Valuation data can be included with the representation. The importance of the memory (positive or negative) can be indicated to a potential autonomous agent through this valuation data. This will enable decisions under the policies: “seek reward”, “avoid punishment”.
- Develop a parallel implementation. The model is particularly fit for parallelization.

6 ETHICAL CONSIDERATIONS

Science in general and research in particular are strongly intertwined with social well-being. It is essential that ethical principles be applied in scientific research. In 1974, the declaration of Helsinki formalized, in the medical field, the ethical principles for research involving human subjects. Since that time, there has been a lot of progress, reflected in high standards that modern society demands [36]. Currently, scientists should know and follow moral principles and practices adequate to every research area such as honesty, carefulness, diligence, openness, fairness, respect and credit [37].

There is common agreement in the scientific community in principles about proper scientific work, scientific integrity, and knowledge production practices that were followed in this research. The methodology and data used in the study are clearly described. The results of the experiments are treated in an objective way and without any pressure at all to arrive to a particular conclusion.

Moreover, there are other indirect ethical issues that could surface from studies like this; whose consequences should not be underestimated. If a theoretical model turns out to be a functional representation of the workings of the brain, concern and excitement would follow, due to the possibilities of using such a critical knowledge to *hack* the brain. Neuromarketing generates concern because it is trying to tailor products or services irresistible to the crowds [38]. What would similar disciplines do if a model can accurately

tell them how a brain would learn or react? On the other side, the development of new techniques for faster and more efficient learning, or theoretical frameworks to understand and overcome cognitive disorders, brings excitement in fields like education, neuromedicine, biomedicine, psychiatry or psychology. At some point, ethics will have to establish standards for these possibilities.

Fuchs [39] reviews ethical problems resulting from brain research which motivated the emergence of a new discipline termed neuroethics. The increasing understanding of brain processes and modifying techniques, challenges essential notions such as free will, agency, moral, judgment, self and personality. Changes in these notions can interfere with essential intuitions about us, inevitably questioning concepts such as responsibility and culpability on which central institutions of our society are based. These questions concerning underlying concepts of humans should be actively dealt with by interdisciplinary debate from disciplines such as philosophy, neuroscience, and humanities.

Neuroeducation [40] is an emerging field that links education and neuroscience in an effort to improve learning and instruction. Neuroeducation reviews key advances in neuroscience, to design training programs of neurocognitive functions, such as working memory, that are expected to have effects on overall brain function. Also, Neuroeducation considers the potential for modern brain imaging methods as diagnostic tools and as measures of the effects of educational interventions.

APPENDIX A SECOND IMPLEMENTATION

Main Classes

```
classdef Hierarchy < handle
    properties
        dimsLevels;
        Thresholds;
        maxSegs;
        Levels;
        numLevels
        inLearning;
        Input
        Output;
    end
    methods
        function obj = Hierarchy(numlevels,dimslevels,thresholds,maxsegs)

        function reset(obj)

        function obj = set_inLearning(obj,val)

        function output = readNext(obj,element)

        function output = readString(obj,string)
    end
end

classdef Level < handle
    properties
        id;
        dimsInput;
        dimsOutput;
        Codeword;
        DFA;
        maxSeg;
        Input;
        Buffer;
        Output;
        inLearning;
    end
```

```
    methods
        function obj = Level(id, dimsInput, dimsOutput, threshold,
                             maxSegment)

        function reset(obj)

        function set_inLearning(obj,value)

        function output = readNext(obj,element)

        function buffer = flushBuffer(obj, element)

        function buffer = returnBuffer(obj)
    end
end

classdef Codebook < handle
    properties
        dimsInput;
        Symbols;
        Codewords;
        Vectors;
        VectorsBinary;
        defaultThreshold;
        Thresholds;
        Input;
        Output;
        inLearning;
    end
    methods
        function obj = Codebook(dimsInput,defaultthreshold)

        function reset(obj)

        function set_inLearning(obj,value)

        function output = readNext(obj,element)

        function winnercodeword = computeOverlapOnInput(obj)

        function symbol = Learn(obj,codeword)

        function [Codeword,CodewordBinary] = getCodeword(obj,symbol)
    end
end
```

```
classdef DFA < handle
    properties
        Transitions;
        Symbols;
        States;
        CurrentState;
        Accepted;
        readNext;
        learnTransition;
        Input;
        Output;
        inLearning;
    end
    methods
        function obj = DFA()

        function reset(obj)

        function set_inLearning(obj,value)

        function [accepted,predictions] = readNext(obj, symbol)

        function destinationState = learnTransition(obj,originalState,
                                    symbol)

        function state = getNewState(obj)

        function state = getAssociatedState(obj,symbol)

        function state = associateNewStateToSymbol(obj, symbol)

        function rewardTransition(obj,state,possymbol)

        function punishTransition(obj,state,possymbol)

        function symbols = getPredictedSymbols(obj,state)
    end
end
```

Example Script

```
% Assumption SeqsBin is a list of sequences in Boolean format

% Hierarchy Parameters
numlevels = 3;
vectorsize = size(SeqsBin{1},2);
dimsLevels = [vectorsize 400; 400 200; 200 100]; % In-out for every Level
thresholds = [0.1 0.25 0.25]; % Similarity thresholds
maxsegs = [50 12 1]; % Temporal parameters

% Hierarchy creation
H = Hierarchy(numlevels,dimsLevels,thresholds,maxsegs);

% Classification
Classes = [];
for s = 1:numel(SeqsBin)
    seq= SeqsBin{s};
    Steps = size(seq,1);
    for j=1:Steps
        output = H.readNext(seq(j,:));
    end
    Classes = [Classes; output];
    H.reset;
end
```

REFERENCES

[1] Daniel J Felleman and David C Van Essen, "Distributed hierarchical processing in the primate cerebral cortex," *Cerebral cortex*, vol. 1, no. 1, pp. 1-47, 1991.

[2] Henry Markram and Rodrigo Perin, "Innate neural assemblies for lego memory," *Frontiers in neural circuits*, vol. 5, 2011.

[3] Van J Wedeen, Douglas L Rosene, Ruopeng Wang, Guangping Dai, Farzad Mortazavi, Patric Hagmann, Jon H Kaas, and Wen-Yih I Tseng, "The geometric structure of the brain fiber pathways," *Science*, vol. 335, no. 6076, pp. 1628-1634, 2012.

[4] Stephen Grossberg, *Adaptive resonance theory*.: Wiley Online Library, 2003.

[5] Jeff Hawkins, *On intelligence*.: Macmillan, 2004.

[6] Ray Kurzweil, *How to Create a Mind: The Secret of Human Thought Revealed*.: Penguin. com, 2012.

[7] Kunihiko Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193-202, 1980.

[8] Maximilian Riesenhuber and Tomaso Poggio, "Hierarchical models of object recognition in cortex," *Nature neuroscience*, vol. 2, no. 11, pp. 1019-1025, 1999.

[9] Jonathan Laserson, "From Neural Networks to Deep Learning: zeroing in on the human brain," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 18, no. 1, pp. 29-34, 2011.

[10] Andrew McAfee and Erik Brynjolfsson, "Big data: the management revolution," *Harvard business review*, vol. 90, no. 10, pp. 60-66, 2012.

[11] Pedro Domingos and Geoff Hulten, "A general framework for mining massive data streams," *Journal of Computational and Graphical Statistics*, vol. 12, no. 4, 2003.

[12] Greg Stuart, Nelson Spruston, and Michael Hèausser, *Dendrites*.: Oxford University Press, 2007.

[13] Vernon Mountcastle, "An organizing principle for cerebral function: The unit model and the distributed system," 1978.

[14] Vernon B Mountcastle, *Perceptual neuroscience: The cerebral cortex*.:

Harvard University Press, 1998.

[15] Donald Olding Hebb, *The organization of behavior: A neuropsychological approach*.: John Wiley & Sons, 1949.

[16] Pentti Kanerva, *Sparse distributed memory*.: The MIT Press, 1988.

[17] Peter Földiak and Malcom P Young, "Sparse coding in the primate cortex," *The handbook of brain theory and neural networks*, vol. 1, pp. 1064-1068, 1995.

[18] Bruno A Olshausen and David J Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?," *Vision research*, vol. 37, no. 23, pp. 3311-3325, 1997.

[19] David J Field, "What is the goal of sensory coding?," *Neural computation*, vol. 6, no. 4, pp. 559-601, 1994.

[20] Peter Földiak, "Forming sparse representations by local anti-Hebbian learning," *Biological cybernetics*, vol. 64, no. 2, pp. 165-170, 1990.

[21] Ke Huang and Selin Aviyente, "Sparse representation for signal classification," in *Advances in neural information processing systems*, 2006, pp. 609-616.

[22] Dileep George, "How the brain might work: A hierarchical and temporal model for learning and recognition," Stanford University, Ph.D. dissertation 2008.

[23] Dileep George and Jeff Hawkins, "Towards a mathematical theory of cortical micro-circuits," *PLoS computational biology*, vol. 5, no. 10, p. e1000532, 2009.

[24] Jeff Hawkins, S Ahmad, and D Dubinsky, "Hierarchical temporal memory including HTM cortical learning algorithms," *Techical report, Numenta, Inc, Palto Alto http://www.numenta.com/htm-overview/education/HTM_CorticalLearningAlgorithms.pdf*, 2011.

[25] Morris W Hirsch, Robert L Devaney, and Stephen Smale, *Differential equations, dynamical systems, and linear algebra*.: Access Online via Elsevier, 1974, vol. 60.

[26] Michael Sipser, *Introduction to the Theory of Computation*.: Thomson Course Technology Boston, 2006, vol. 2.

[27] Anil K Jain, Robert P. W. Duin, and Jianchang Mao, "Statistical pattern recognition: A review," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 1, pp. 4-37, 2000.

[28] Satosi Watanabe, *Pattern recognition: human and mechanical*.: John Wiley & Sons, Inc., 1985.

[29] John Medina, *Brain rules: 12 principles for surviving and thriving at work, home, and school*.: Pear Press, 2010.

[30] HB Barlow, "Cerebral cortex as model builder," *Models of the visual cortex*, pp. 37-46, 1985.

[31] Douglas R Hofstadter, "Analogy as the core of cognition," *The analogical mind: Perspectives from cognitive science*, pp. 499-538, 2001.

[32] Jeff Hawkins, Dileep George, and Jamie Niemasik, "Sequence memory for prediction, inference and behaviour," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 364, no. 1521, pp. 1203-1209, 2009.

[33] John Wright, Yi Ma, Julien Mairal, Guillermo Sapiro, Thomas S Huang, and Shuicheng Yan, "Sparse representation for computer vision and pattern recognition," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1031-1044, 2010.

[34] CMU. (2003) Carnegie-Mellon MoCap Database. [Online]. http://mocap.cs.cmu.edu

[35] Loc Huynh, Thanh Ho, Quang Tran, Thang Ba Dinh, and Tien Dinh, "Robust classification of human actions from 3D data," in *Signal Processing and Information Technology (ISSPIT), 2012 IEEE International Symposium on*, 2012, pp. 263-268.

[36] WMA, *World Medical Association Declaration of Helsinki: ethical principles for medical research involving human subjects*.: World Medical Association, 2008.

[37] D. Resnick. (2011) National Institute of Environmental Health Sciences. [Online]. http://www.niehs.nih.gov/research/resources/bioethics/whatis/index.cfm

[38] Dy Bridger D Lewis and D Phil, "Market researchers make increasing use of brain imaging," *Nature Neuroscience*, vol. 7, no. 7, p. 683, 2004.

[39] Thomas Fuchs, "Ethical issues in neuroscience," *Current opinion in Psychiatry*, vol. 19, no. 6, pp. 600-607, 2006.

[40] Daniel Ansari, Bert De Smedt, and Roland H Grabner, "Neuroeducation - a critical overview of an emerging field," *Neuroethics*, vol. 5, no. 2, pp. 105-117, 2012.