ESTUDIO TEÓRICO-EXPERIMENTAL DE UN MODELO MATEMÁTICO DE LA MEMORIA EPISÓDICA EN UN ROBOT MÓVIL

THEORETICAL-EXPERIMENTAL STUDY OF A MATHEMATICAL MODEL OF THE EPISODIC MEMORY IN A MOBILE ROBOT

GUILLERMO MARTÍN SÁNCHEZ

TRABAJO DE FIN DE GRADO DEL DOBLE GRADO EN INGENIERÍA INFORMÁTICA - MATEMÁTICAS

FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

https://github.com/guillemarsan/Mathematical-model-of-episodic-memory

---

18-09-2020

Director:

JOSÉ ANTONIO LÓPEZ OROZCO

Resumen en castellano

La memoria episódica consiste en la habilidad de recordar de manera consciente episodios sucedidos al individuo. Está basada en la memoria asociativa, capacidad de agrupar estímulos que ocurren de forma simultánea, formando así conceptos. Estudiamos un modelo de la memoria asociativa basado en la modelización del hipocampo con una red neuronal. Exploramos y solucionamos los problemas que surgen al implementar el modelo para usarlo para asociar estímulos complejos. Finalmente, usamos este modelo del hipocampo en el algoritmo de control de un robot móvil para la resolución de problemas tanto en un entorno simulado como en la vida real para mostrar su utilidad como sistema de memoria artificial.

Palabras clave

Hipocampo, Memoria asociativa, Memoria episódica, Neurona conceptual, Robot móvil, Visión por ordenador.

Abstract

The episodic memory consists on the ability to consciously remember episodes experienced by the individual. It is based on the associative memory, capacity to group together stimuli perceived simultaneously, creating concepts. A model of the associative memory based on the modelisation of the hippocampus with a neural network is studied. Problems related to the implementation of the model for complex stimuli association are explored and solved. Finally, this hippocampus model is used in the control algorithm of a mobile robot for problem solving both in a simulated and a real environment, to show its utility as an artificial memory system.

Keywords

Associative memory, Computer Vision, Concept neuron, Episodic memory, Hippocampus, Mobile robot.

Table of Contents

Table of symbols	
Symbol	Usage
k	either a scalar ($k \in \mathbb{R}$) or a function $k(t) : \mathbb{R} \to \mathbb{R}$
$\boldsymbol{k}$	either a vector ($\boldsymbol{k} \in \mathbb{R}^m$) or a function $\boldsymbol{k}(t) : \mathbb{R} \to \mathbb{R}^m$
$\boldsymbol{K}$	either a matrix ($\boldsymbol{K} \in \mathbb{R}^{m \times n}$) or a function $\boldsymbol{K}(t) : \mathbb{R} \to \mathbb{R}^{m \times n}$
$\dot{x}$	derivative of $x(t)$
$\langle \cdot, \cdot \rangle$	euclidean inner product
$\|\cdot\|$	euclidean norm
$\mathcal{U}(A)$	uniform probability distribution in set A
$\mathcal{P}(A)$	probability of A
$\mathcal{P}(A\|B)$	probability of A given B
n	dimensionality of the stimuli
L	number of stimuli
M	number of neurons in the first layer
N	number of neurons in the second layer
K	number of stimuli associated together
$\boldsymbol{x}$	stimulus vector
$\boldsymbol{s}_{sel}(t)$	input function to the first layer
$\boldsymbol{W}$	weights matrix of the first layer
$\boldsymbol{y}(t)$	activation of the first layer
$\boldsymbol{s}_{con}(t)$	input function to the second layer
$\boldsymbol{U}$	weights matrix of the second layer
$\boldsymbol{y}_{con}(t)$	activation of the second layer
T	total training time
t_σ	time each stimulus is presented

Table of abbreviations	
Abbreviation	Usage
MTL	Medial Temporal Lobe
ReLU	Rectified Linear Unit
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
ResNet	Residual Network

Chapter 1

Introduction

1.1 Background

In 1966, R. Bellman firstly discussed and coined the term 'curse of dimensionality' in his famous work *'Dynamic programming'*[1]. He talked about the arising difficulty high dimensional spaces suppose to optimization. Since then this term has been used to denote several more undesired characteristics of these spaces which include the combinatorial exponential growth and the need of more training data in machine learning tasks. To avoid these undesired effects several techniques of dimensionality reduction such as principal component analysis (PCA) or, more recently, autoencoders[2], are daily used to ensure good model performance.

It is under this circumstances that some articles where published stating that the high dimensional nature of their problems was also key to finding an optimal solution for their optimization tasks[3]. The term 'blessing of dimensionality' was coined and considered as the other side of the coin by D. L. Donoho in his data science Manifesto: *'High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality'*[4]. Several applications such as face recognition[5], multidimensional cluster analysis[6] or error correction in machine learning systems[7] used the term to explain their good performance.

Scientists concluded that dimensionality was a curse when not fully understood but a blessing when fully exploited. As A. N. Gorban and I. Y. Tyukin concluded in their paper abstract[8] in 2018: "At the beginning of the twenty-first century, it became clear that the proper utilisation of these phenomena in machine learning might transform the curse of dimensionality into the blessing of dimensionality".

The brain is high dimensional[9]. It consists of billions of neurons working as coupled dynamical systems to form high dimensional representations of the world for problem solving. Another point of view in which scientists have been paying attention to this high dimensionality lately is the number of synaptic inputs a particular neuron has. A single rat hippocampal CA1 pyramidal cell can be post-synaptic to up to around 30,000 excitatory and 1700 inhibitory synapses[10]. In humans, there is experimental evidence to believe that the variation in number of synaptic connections could explain 25% of intelligence quotient

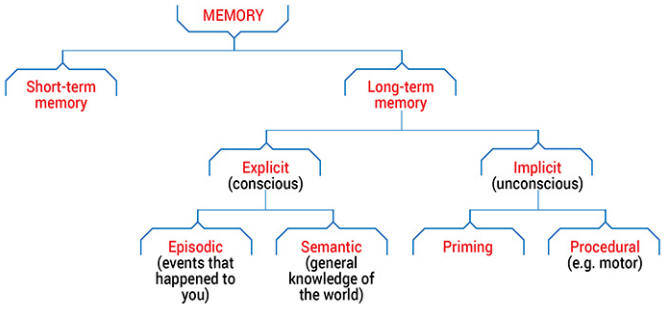

Figure 1.1: Types of memory[17]. The focus of our work is in the formation of episodic memories through associative memory.

(IQ) scores variance[11].

I. Y. Tyukin *et al.*, researchers of the research group Cognitive Systems and Neurorobotics of the Faculty of Mathematics at Universidad Complutense de Madrid, first study the relationship between the blessings of dimensionality (high dimensional points generated stochastically with a distribution with bounded support are with high probability Fischer-separable) and concept cells[12]. This leads to their most recent work where they propose a neuronal model for associative memory and test its feasibility with constrained examples: stimuli sampled from a uniform distribution in the hypercube $([0,1])^n$ and simple intensity wave input. In doing so they also provide theoretical justification of the existence of concept cells based on the high neuronal dimension as the major factor[13].

In 1972, Barlow said: "our perceptions are caused by the activity of a rather small number of neurons selected from a very large population of predominantly silent cells."[14] in a manifest supporting the idea of concept cells. This was contrary to the previous belief that a single or few cells could not encode abstract concepts but rather, the orchestrated activity of a large group of neurons could. In 2005, R. Quian Quiroga *et al.* showed experimental evidence of the existence of these concept cells[15]. Intracranial electrodes recorded brain activity of the medial temporal lobe (MTL) of subjects while performing different tasks. They discovered an specific neuron that would fire to 7 different pictures of Jennifer Aniston but not pictures of other famous people or places. Furthermore, they showed this neuron would fire when the subject read the written or heard the spoken name of the actress. This experimental evidence was remarkably important in the single-cell doctrine which lead to people start calling the previously called 'grandmother cells' (term coined to ridicule the idea), 'Jennifer Aniston cells'. More recently, 'conceptual cells' has been established as preferred term. In 2012, R. Quian Quiroga wrote about how these neurons could be understood to create and retrieve associations between stimuli of different nature (e.g. visual input as an image and written language, and auditory input as spoken language) to form the "building blocks of episodic memory"[16].

Memory can be classified in different types as show in Fig. 1.1. Episodic memory consists on the ability to retrieve past events that an individual has lived. 'Associative learning' and 'associative memory' are terms used to refer to the integration, storage and later retrieval

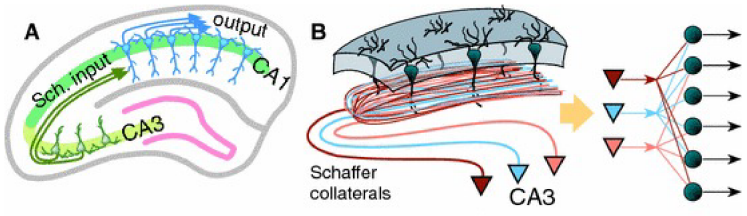

Figure 1.2: [12] A) Layered structure of the hippocampus leads to parallel connections between the CA3 and CA1 layers. B) Via Schaffer collaterals the connections are dense: the axons of each of the CA3 pyramidal neurons reach the dendrites of many CA1 neurons leading to a convergence-divergence of information.

of associated signals in the brain[18], ability used to later form episodic memory. The main way in which two stimuli are associated in the brain is through time simultaneity. In this way an event could be understood to be constructed through the association of stimuli that happened at the same (or closely in) time.

The hippocampus (which is part of the MTL with the parahippocampal and perirhinal cortices) is considered the core of episodic memory in the human brain. This is why I. Y. Tyukin *et al.* use the mathematical modelisation of the CA3 and CA1 regions of this part of the brain for their work on concept cells and their role in associative memory. Pyramidal cells in CA1 are parallel oriented to the same axis. They receive excitatory input from the CA3 neuron's axons that bifurcate and cross CA1 in parallel leading to high connectivity between each CA3 neuron and many CA1 neurons. This allows the passing of high dimensional data in a convergence-divergence manner (see Fig. 1.2).

We are seeing deep neural networks complete numerous pattern recognition tasks successfully. However, they still perform poorly in cognition benchmarks[19]. In the same way first artificial neural networks (ANNs) where invented taking inspiration from biological systems it seems clear that the step forward is to keep using new research in neuroscience to develop new machine learning algorithms. As H. Sinz *et al* put in his manifesto *'Engineering a Less Artificial Intelligence'*[20]: "Despite enormous progress in machine learning, artificial neural networks still lag behind brains in their ability to generalize to new situations" and "neuroscience can guide the quest for better inductive biases by providing useful constraints on representations and network architecture". They focus on the well-known idea that while ANNs need thousand of examples to learn, the brain generalizes well with many fewer examples.

There has been attempts to modelise biological systems more closely to solve some of these problems. Spiking neural networks (SNNs)[21] use neurons that mimic the biological ones in greater detail: they use some time dependent differential equation, e.g. the Hodgkin–Huxley model[22], to modelise the membrane potential of each neuron at each time step. Experimental evidence suggests that biological neural systems use the timing of single action potentials to encode information. However, this approach is computationally very expensive. Another modelisation is the Adaptive resonance theory (ART)[23] which has lead to a number of different neural network algorithms with the main idea that changes in the

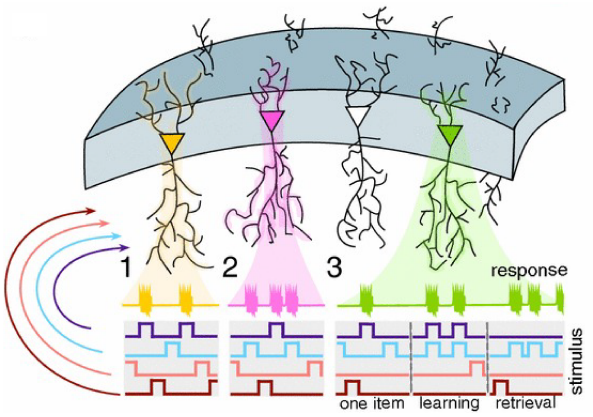

Figure 1.3: Different types of neurons. 1) A neuron is selective when it fires only for a specific stimulus. We expect neurons of the first layer to become selective. 2) A neuron is conceptual when it fires whenever any of a set of associated stimuli are presented. We expect neurons of the second layer to become conceptual. 3) The association of stimuli is done through learning: stimuli that are presented at the same time are associated together.

weights do only occur when the difference between the sensation and the expectation overcome a 'vigilance parameter'. In this way, they modelise the brain active part in perception. It is justified that this kind of networks overcome the incremental learning problem, i.e. the problem of acquiring new knowledge without destroying the previous knowledge stored. The architecture of these networks is very similar to the associative memory model we are going to study in this project but they use a top-down algorithm instead of a more distributed biological plasticity rule.

The associative memory or hippocampus model consists of a two layer neural network that modelise the CA3 and CA1 layers. Through Hebbian learning we expect the neurons in the first layer to become selective, that is, each neuron fires to a single stimuli. We also expect the neurons in the second layer to become concept neurons, that is, each neuron fires whenever one of a number of stimuli that form the concept is used as input. The association of stimuli into concepts is done through unsupervised learning. However, differently from usual unsupervised methods, stimuli are not grouped depending on the similarity of their features but rather if they are shown to the network close in time simulating in this way associative memory. This allows for very different stimuli, even in nature (as pointed out before between visual and auditory signals) to form a single concept as long as they are time related. These ideas are represented in Fig. 1.3.

The present project tries to explore the hippocampus model experimentally to see its feasibility as an algorithm for artificial cognition. We do not expect to perform better than current techniques for machine learning classification but rather explore a different way of solving the same task in a more biologically inspired manner.

1.2 Objectives

The student was assigned a Collaboration Scholarship to aid with research in the Cognitive Systems and Neurorobotics research group, and in particular, further develop the insight into this model.

In a previous work which constitutes the student's Bachelor's Project of Mathematics, the mathematical foundations and justification for the model are explored. A series of novel modifications to fix a particular issue are proposed and theoretically and numerically tested to choose the best option.

As a next step, we explore in this project the improved model outside of the theoretical domain and in practice. In order to justify the model experimentally and to show its application in artificial cognition, an exploration of its capabilities as a tool in real life circumstances is needed. This includes not only the complexity of the stimuli associated but also nuances that come from sampling, predicting, time constraints, etc.

Our main objective is to use the model inside a mobile robot to experimentally show the feasibility of its use as a practical tool for problem solving (giving it as well some validation as a model for the hippocampus). As a previous necessary instrumental goal, we want to implement the model and train it on natural stimuli. Along the way we learn the main constraints of the learning process, parameter tuning and overall test its ability to correctly associate stimuli more complex than the previously explored.

Therefore the questions we expect to answer are:

- Is the model able to associate natural stimuli? What are the main constraints to take into account or to solve?
- Can a mobile robot use the model for a significant application?

1.3 Approach

Firstly, to achieve the preliminary goal, we implement the model in MATLAB and test it for increasingly complex stimuli. We program it in MATLAB since it is the programming language the research group uses for implementing algorithms and to control the robot. Our final goal is to use a webcam attached to the robot for stimuli sampling so we investigate the limits of the model that give constraints to the stimuli in order to later design the experiment. Our focus is less in modifying the model but in studying how complex the stimuli can be and what problems arise with this complexity.

Secondly, taking into account the constraints previously explored, we study the constraints the robot capabilities also set for the experiment design. The research group Cognitive Systems and Neurorobotics owns a Pioneer 3DX mobile robot. They have already used it in their previous works. It is a differential wheeled robot: two motored wheels allow it free movement. It also has sonars which can be used for navigation. A webcam on top of a supportive base has been added to allow it to visually explore the environment. A gripper can also be easily installed.

Thirdly, and with all of this taken into account, we design our experiments. In them we expect the robot to use its associative memory to solve a series of tasks. We use the constraints about the stimuli learned in the first phase and the constraints about the robot learned in the second phase to design the experiments.

Finally we design and program the robot's control algorithm (which includes the associative memory model) and test how well it solves the expected tasks.

Due to the 2020 SARS-CoV-2 pandemic, the accessibility to the robot stopped being possible and therefore we had to simulate the experiment. Since the model was implemented in MATLAB and the robot worked executing MATLAB scripts, we decided to use Simulink, a graphical programming environment based on MATLAB code for programming dynamic systems, to program the robot's control algorithm and environment simulation. We found there was already a project which constituted the basic skeleton for simulation of the Pioneer 3DX in Simulink[24] and decided to work from there. In order to make the work productive and in regards of the future testing on the robot, we programmed the robot's algorithm trying to simulate as possible the actual architecture and specifications of the Pioneer, and use the modular nature of Simulink to make the interchangeability from a simulated environment to an actual real life experiment as direct as possible. In the end, we were able to access the robot and so we used this direct interchangeability to execute a toy experiment, to show that indeed the robot's control algorithm also works in real life situations.

1.4 Overview

In this Chapter we have presented the motivation and objectives to the present project as well as our approach to reach them. In Chapter 2 we present the hippocampus mathematical model and we explain it in great detail. Secondly, in Chapter 3 we describe the process of implementation and testing of the model. We discuss the problems encountered, how we solved them and the final results. In Chapter 4 we explore the constraints of the robot, design the experiments, explain the robot's control algorithm and environment simulation and present our final results. Finally, in Chapter 5, we discuss the conclusions and talk about possible future work.

Chapter 2

Model

2.1 Architecture

The model consists of a two layer neural network. The first layer (called 'selective layer'), consisting of M neurons, receives the L nD (n-dimensional) stimuli $\boldsymbol{x}_i \in \mathbb{R}^n$. The input function that determines at time t the input to the first layer is:

$$\boldsymbol{s}_{sel}(t) = \sum_{i=1}^{L} \sqrt{\frac{3}{n}} \boldsymbol{x}_i \sigma_i(t), \tag{2.1}$$

where σ_i are disjoint rectangular time windows that determine when stimulus $\boldsymbol{x}_i$ is presented. In general, we will be showing each stimulus for t_σ units of time cyclically for a maximum time T.

The mD activation of the first layer to stimulus i given by $\boldsymbol{y}_i \in \mathbb{R}_+^m$ is used as input for the second layer (called 'concept layer') with N neurons. Due to short-time memory, implemented in biological systems through e.g. synaptic integration, K consecutive signals $\boldsymbol{y}_i$ can overlap in time. Therefore, the input function to the second layer is:

$$\boldsymbol{s}_{con}(t) = \sum_{i=1}^{K} \boldsymbol{y}_i \chi_i(t), \tag{2.2}$$

where χ_i are overlapping rectangular time windows. If the learning is done with $t \in [0, K\Delta]$, then $\chi_i(t) = 1$ when $t \in [(i-1)\Delta, K\Delta)$ and $\chi_i(t) = 0$ otherwise. This means that the synaptic integration is done such that the signals overlap as follows:

$$\{\boldsymbol{y}_1\}, \{\boldsymbol{y}_1, \boldsymbol{y}_2\}, ..., \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_K\}, \{\boldsymbol{y}_{K+1}\}, \{\boldsymbol{y}_{K+1}, \boldsymbol{y}_{K+2}\}, ..., \{\boldsymbol{y}_{K+1}, \boldsymbol{y}_{K+2}, ..., \boldsymbol{y}_{2K}\}, \{\boldsymbol{y}_{2K+1}\}, ... \tag{2.3}$$

The equations for the j-th neuron activation in the selective layer are:

$$v_j(t) = \langle \boldsymbol{w}_j(t), \boldsymbol{s}_{sel}(t) \rangle, \qquad y_j(t) = H(v_j(t) - \theta_j), \tag{2.4}$$

where $\boldsymbol{w}_j(t) \in \mathbb{R}^n$ is the vector of synaptic weights of the nD stimulus to the j-th neuron, $\langle \cdot, \cdot \rangle$ is the usual inner product, θ_j is the threshold of activation (in general we will consider

$\theta_j = \theta \ \forall j$) and $H(u) = max(0, u)$, i.e. the usual rectifier activation function for rectified linear units (ReLU).

2.2 Learning

The learning differential equation simulates the Hebbian synaptic plasticity encountered in biological systems:

$$\dot{\boldsymbol{w}}_j(t) = \alpha y_j(t)(\beta^2 \boldsymbol{s}_{sel}(t) - v_j(t)\boldsymbol{w}_j(t)). \tag{2.5}$$

where $\alpha > 0$ is the learning parameter and $\beta > 0$ is an order parameter that we will define later.

Similarly to Oja's rule[25], we use the term $y_j \boldsymbol{s}_{sel}$ to evoke plastic changes only when the input generates neuronal response. The term $-v_j \boldsymbol{w}_j$ ensures boundness of $\|\boldsymbol{w}_j\|$ for convergence and model physiological boundaries.

The activation equations and learning rules for the second layer are analogous to those presented previously but with input function $\boldsymbol{s}_{con}$ and their own constants θ_{cn} and β_{cn}.

To integrate the learning equation Eq. 2.5 we initialise for every neuron (we drop the index j in the following) its synaptic weights randomly uniformly in the hypercube with side length 2, i.e. $\boldsymbol{w}_0 \sim \mathcal{U}([-1, 1]^n)$. Then we use a predictor-corrector method with the explicit Euler method and the trapezoidal rule for numerical integration:

$$\begin{cases} v_i = \langle \boldsymbol{w}_i, \boldsymbol{s}_{sel}(t_i) \rangle \\ y_i = H(v_i - \theta) \\ \boldsymbol{dw}_i = \alpha y_i(\beta^2 \boldsymbol{s}_{sel}(t_i) - v_i \boldsymbol{w}_i) \\ \widetilde{\boldsymbol{w}}_{i+1} = \boldsymbol{w}_i + h\boldsymbol{dw}_i & (pred) \\ & i = 0, 1, ..., T/h - 1 \\ \widetilde{v}_{i+1} = \langle \widetilde{\boldsymbol{w}}_{i+1}, \boldsymbol{s}_{sel}(t_{i+1}) \rangle \\ \widetilde{y}_{i+1} = H(\widetilde{v}_{i+1} - \theta) \\ \widetilde{\boldsymbol{dw}}_{i+1} = \alpha \widetilde{y}_{i+1}(\beta^2 \boldsymbol{s}_{sel}(t_{i+1}) - \widetilde{v}_{i+1}\widetilde{\boldsymbol{w}}_{i+1}) \\ \boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \frac{1}{2}h(\boldsymbol{dw}_i + \widetilde{\boldsymbol{dw}}_{i+1})) & (corr) \end{cases} \tag{2.6}$$

Let $d_j \in \{0, 1, ..., L\}$ the number of stimuli the j-th neuron fires to. We will say the neuron is *inactive* if $d_j = 0$ and *active* otherwise. Furthermore, if $d_j = 1$ we will say the neuron is *selective*. If a stimulus does not make any neuron fire we will say it is *lost*.

Let's see how learning can improve the number of selective neurons. Focusing on the j-th neuron, we denote by $\boldsymbol{h} = \sqrt{\frac{3}{n}}\boldsymbol{x}_i$ the first stimulus that activates it at time $t = t^*$ ($y_j(t < t^*) = 0$, $y_j(t^*) > 0$). Denoting $\boldsymbol{w} = \boldsymbol{w}_j$, we split it into the parallel and orthogonal to $\boldsymbol{h}$ components as follows: $\boldsymbol{w} = \boldsymbol{w}_\parallel + \boldsymbol{w}_\perp$. Note $\langle \boldsymbol{w}_\parallel, \boldsymbol{w}_\perp \rangle = 0$ and we can express $\boldsymbol{w}_\parallel := q(t)\frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}$. Then, Eq. 2.5 yields:

$$\begin{aligned} \dot{\boldsymbol{w}}_\perp &= -\alpha \|\boldsymbol{h}\| y q \boldsymbol{w}_\perp, \\ \dot{q} &= \alpha \|\boldsymbol{h}\| y(\beta^2 - q^2). \end{aligned} \tag{2.7}$$

Since $v(t^*) = \langle \boldsymbol{w}, \boldsymbol{h} \rangle = \langle \boldsymbol{w}_{\parallel}, \boldsymbol{h} \rangle + \langle \boldsymbol{w}_{\perp}, \boldsymbol{h} \rangle = q(t^*)\|\boldsymbol{h}\|$ and $y(t^*) > 0$ we have $v(t^*) = q(t^*)\|\boldsymbol{h}\| > \theta \Rightarrow q(t^*) > \theta/\|\boldsymbol{h}\|$. So, from Eq. 2.7 if $\beta > \theta/\|\boldsymbol{h}\|$ then $\dot{q} > 0$ and thus we ensure $y(t \geq t^*) > 0$. Furthermore, we get $\boldsymbol{w}_{\perp} \to 0$ and $q \to \beta$, which implies that:

$$\lim_{t\to\infty} \boldsymbol{w}(t) = \beta \frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}. \tag{2.8}$$

We conclude that if α is sufficiently large, during learning the synaptic weights "align" along the stimulus $\boldsymbol{h}$ and the neuron becomes active when this stimulus is shown. More importantly, for high n and sufficiently different (orthogonal) stimuli we get $\langle \boldsymbol{h}, \boldsymbol{x}_k \rangle \leq 0 \quad \forall k \neq i$ and thus the neuron becomes selective to this stimulus.

2.3 Inhibition

Numerically we observe neurons in the selective layer align too quickly to the few first stimuli shown. This leads to a high number of selective neurons for the first stimuli but results in lost stimuli (stimuli to which no neuron fires) for the last stimuli. To avoid this, as previous work not contemplated in this project, we proposed and studied novel modifications of the learning rule which focused in different ways to implement inhibition along the neurons of the same layer. In particular we decided to use the one that showed better results. The modified learning rule adds inhibition as follows:

$$\begin{cases} v_i^{in} = d \sum_{j=1}^{M} y_i^k (y_i^k - y_i^j) \\ y_i^{in} = H(v_i^{in} - \theta_{in}) \\ v_i = \boldsymbol{w}_i \boldsymbol{s}(t_i) - 1 * y_i^{in} \\ y_{i+1} = H(v_i - \theta) \end{cases} \qquad i = 0, 1, ..., T/h - 1 \tag{2.9}$$

where $d \in \mathbb{R}$ is a parameter to fit. Experimentally we see that good results (low lost stimuli and high selectivity) are achieved with $d = 150$.

This inhibition factor can be vaguely understood as adding a supportive layer of inhibitory neurons to each of the layers, with the same number of neurons respectively (see Fig. 2.1). Each one of them inhibits the activation of one of the neurons and receives excitatory input from each of the other neurons (except the ones they inhibit). When several neurons activate for the same input, the one with stronger activation 'silences' the rest leaving them free to align to later inputs. The excitatory synaptic weight between the neurons of the model and the inhibitory ones as well as the inhibitory synaptic weight between the inhibitory neurons and the neurons of the model stay constant (1 and -1 respectively) i.e. they do not go through Hebbian plasticity for simplification.

2.4 Retrieval

Memory retrieval is done via input of the stimuli perceived $\boldsymbol{s}_i$ to the neural network. Once the synaptic weights between the input and the first layer $\boldsymbol{W}$ and the ones between the first

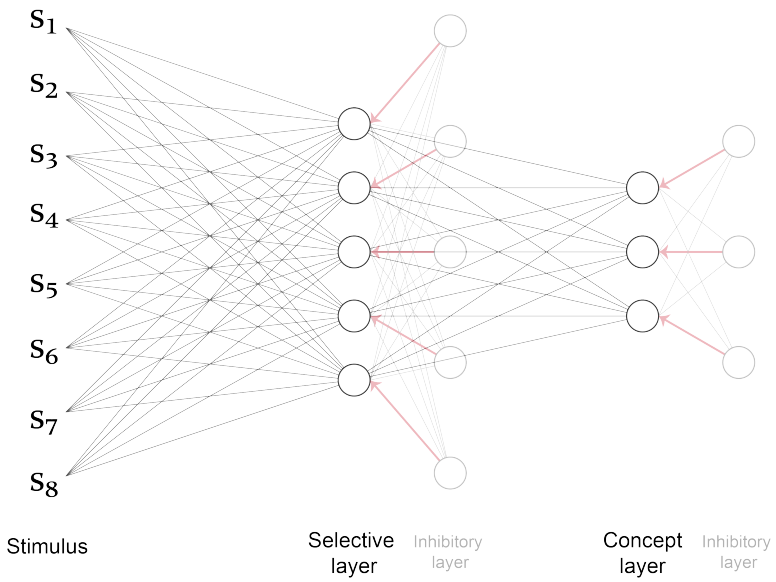

Figure 2.1: Architecture with inhibition layers. In black the model neurons, in grey the inhibitory neurons. The activation is forwards except when shown by arrows. Light red represent constant inhibitory connections (weights = -1) and light grey constant excitatory connections (weights = 1).

layer and the second $\boldsymbol{U}$ are computed through learning, we compute the activation of the last layer neurons as follows:

$$\begin{aligned} \boldsymbol{v} &= \langle \boldsymbol{W}, \boldsymbol{s_i} \rangle, \qquad & \boldsymbol{y} &= H(\boldsymbol{v} - \theta), \\ \boldsymbol{v}_{cn} &= \langle \boldsymbol{U}, \boldsymbol{y} \rangle, \qquad & \boldsymbol{y}_{cn} &= H(\boldsymbol{v}_{cn} - \theta_{cn}). \end{aligned} \tag{2.10}$$

During the retrieval the Hebbian plasticity stops and we do not compute the effect of the inhibitory neurons: we consider it a separate task from learning.

2.5 Summary

The neural network consists of two layers of ReLUs. During the learning phase we use the differential equation Eq. 2.5 to update the weights of both layers. We use inhibition during learning to ensure there are fewer lost stimuli. During a second phase of retrieval we compute the activation of the layers through forward propagation without plastic changes and without using inhibition (see Fig. 2.2).

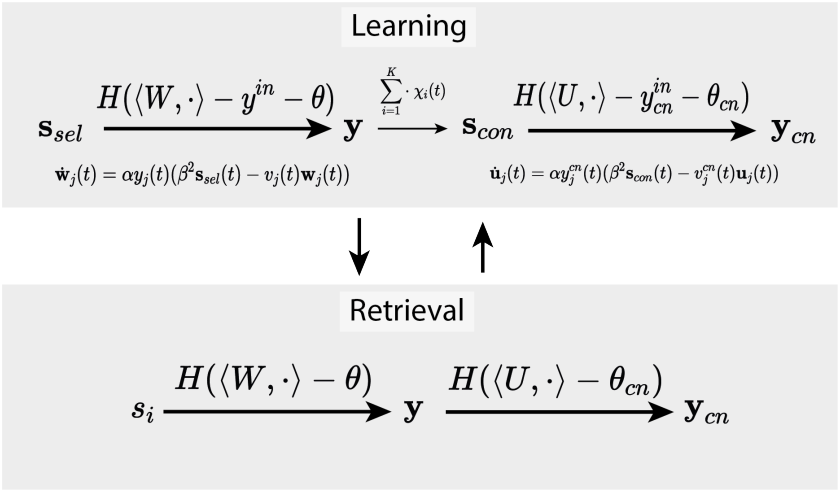

Figure 2.2: Diagram of the two states of the system. During learning we use plasticity, synaptic integration and inhibition. During retrieval we only compute the activation through forward propagation.

With learning we expect the first layer to become highly selective with not many lost stimuli, i.e. it becomes a layer able to recognise all stimuli. We also expect the second layer to become conceptual to stimuli showed closely in time (all K stimuli in a concept), i.e. it becomes able to associate stimuli that happened close in time together, implementing associative memory, the building block of episodic memory.

Chapter 3

Implementation of the model

3.1 First steps

Since for the concept layer the nature of the stimuli is nonimportant, as long as the selective layer exhibits high selectivity, we implemented a simplified model consisting only of the first layer and tested it with stimuli of different nature. In particular, we started with simple stimuli but more complex than the already tested (random uniform sample in $[-1, 1]^n$ and simple oscillation functions). We tested it with binary simple line patterns and with the MNIST library[26].

3.1.1 Patterns and the problem of similarity

We first test the first layer on a series of 7×7 binary simple patterns consisting of all the vertical, horizontal and diagonal lines. Some of these patterns can be seen in Fig. 3.1.

We used the $n = 7^2 = 49$ binary vectors resulting of linearising $L = 40$ patterns as stimuli. We use a layer with $M = 100$ neurons. The parameters used where the ones suggested by the paper which, even though based on the no longer true assumption that the stimuli are randomly sampled form a uniform distribution, turn out to perform well for this case. In particular we set $\theta = \frac{\sqrt{3}}{2}$ and setting $p_{sl} = 0.95$ we compute β as follows:

$$\beta = \frac{\theta}{\delta}, \qquad \delta = \sqrt{1 - \frac{2\Phi^{-1}(p_{sl})}{\sqrt{5n}}}, \tag{3.1}$$

where Φ is the normal cumulative function (for details reference back to the paper[13]). We train for $T = 400$ showing each stimulus cyclically for $t_\sigma = 0.5$.

The results achieved are shown in Fig. 3.2. On the top we see a rasterplot that represents which stimuli (rows) make each neuron (columns) fire. The columns are ordered for visualization. We see that most of the neurons are selective (columns with only one row in black) and that most of the stimuli are not lost (rows with some column in black). On the bottom, we see the results numerically: for each neuron we plot how many stimuli make it fire. Notice that in this case the neurons are not ordered for visualization as in the top

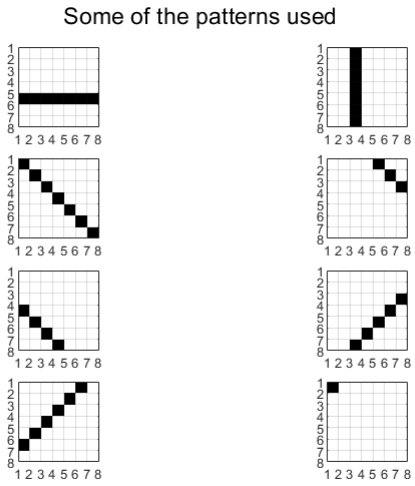

Figure 3.1: Sample of the 7×7 simple binary patterns used: vertical, horizontal and diagonal lines.

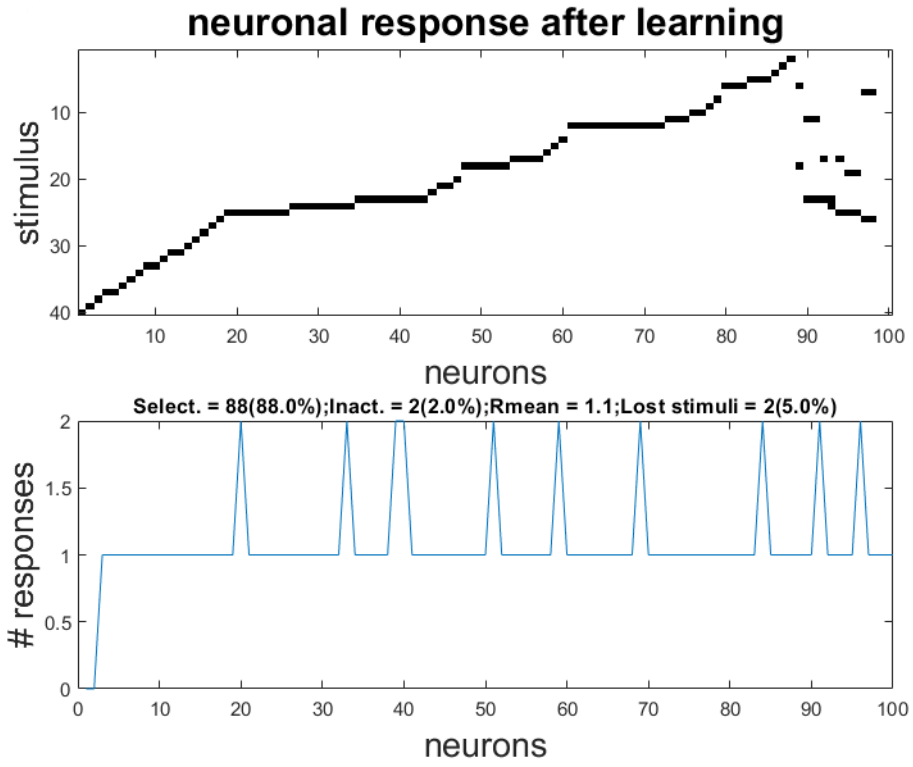

Figure 3.2: Results of the selective layer with simple patterns. Top: Rasterplot showing the activation of neurons to a given stimulus (neurons ordered for visualization). Bottom: Graph that shows the number of responses for each of the neurons (not ordered). We see high selectivity and low inactivity. We measure few lost stimuli.

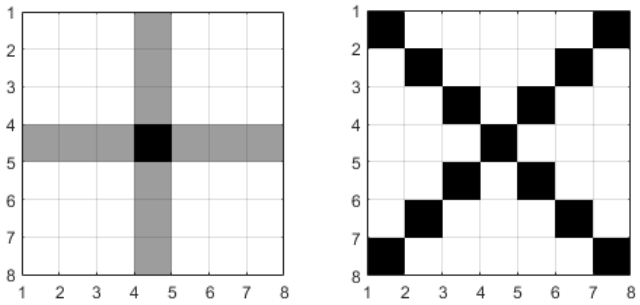

Figure 3.3: Crosses patterns added to the training set to test the ability of the selective layer to work with more complex patterns.

graph. In particular, the ones with 0 responses are inactive neurons and the ones with 1 response are selective. We observe, as the rasterplot suggested, that we achieve a 88% of selectivity. We also measure only 2 (5%) lost stimuli.

We later decided to add two more complex patterns (a vertical and diagonal cross) to the training set (see Fig. 3.3). We saw the first problem arise.

As the theory determines, we need the stimuli to be orthogonal enough so that when the weight vector $\boldsymbol{w}$ of a neuron aligns to one of the stimuli it stops firing for the rest. This is key for the first layer to become selective to each of the stimuli and to be able to distinguish between them. When stimuli are not sufficiently different (orthogonal) the first layer fails to become selective: no selective neurons arise for this set of similar stimuli. Furthermore, some neurons align such that they become active for all the stimuli in the set. This later leads to problems for the conceptual layer to learn the association and therefore should be avoided. We will call this the *problem of similarity*.

When adding the crosses we encounter this problem as shown in Fig. 3.4. In our particular case we can see the problematic patterns in Fig. 3.5. We see they are a pair in which pattern 15 is contained inside pattern 42. In particular, the cosine of the angle they form is 0.7338. This makes them too similar. In comparison, the highest cosine the pattern 15 forms with any other stimuli is 0.378.

We measure how similar two stimuli are by computing the cosine of the angle their vector representations form as follows:

$$cos(\theta) = \frac{\langle \boldsymbol{s}_1, \boldsymbol{s}_2 \rangle}{\|\boldsymbol{s}_1\| \|\boldsymbol{s}_2\|}. \tag{3.2}$$

3.1.2 Digits and the problem of difference

In this case we used the digits of the MNIST database as stimuli. The MNIST database is a collection of 60.000 examples of 20×20 greyscale images of handwritten digits. A sample of these digits can be seen in Fig. 3.6. It is commonly used as a benchmark for testing classification algorithms. We will use it to test if the first layer becomes selective with more

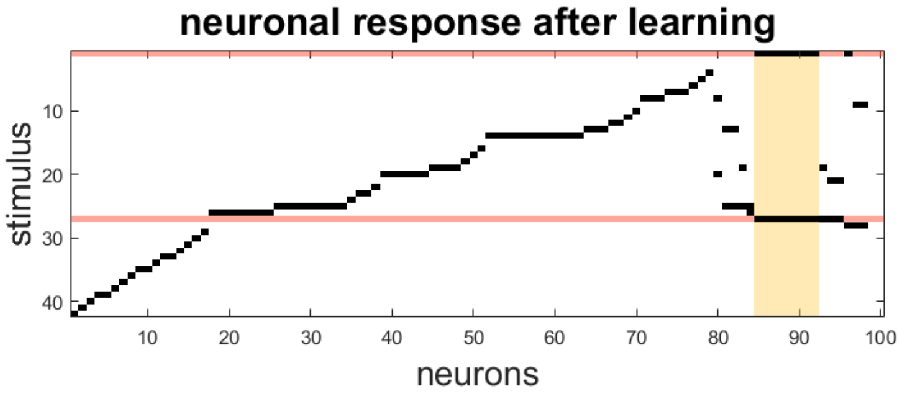

Figure 3.4: Rasterplot showing the neuronal response when the crosses are added. In red the stimuli (pattern 15 and pattern 42) that are similar. We see there are no selective neurons for these two stimuli. In yellow neurons that behave like concept neurons: they fire for both stimuli.

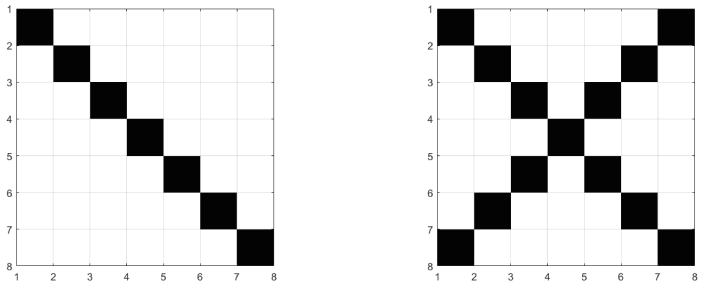

Figure 3.5: Similar patterns that generate *the problem of similarity*. Pattern 15 (left) and pattern 42 (right).

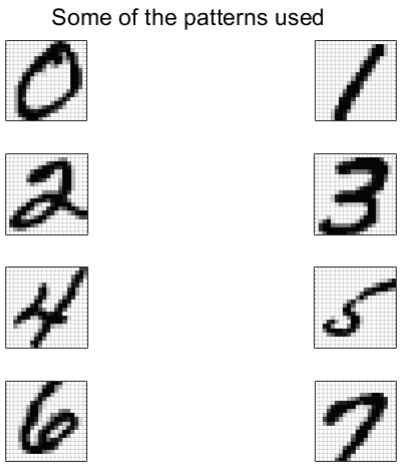

Figure 3.6: Sample of the MNIST database.

complex inputs and how similar a new stimulus has to be to the one presented during the learning process to be identified as the same one.

We use the same training parameters as before, except in this case we have $L = 20$ greyscale vectors with $n = 20^2 = 400$ components. Due to the increase in dimensionality we now use $M = 200$ neurons in the selective layer.

The results achieved are shown in Fig. 3.7. No stimuli are lost and we achieve 113 (56.5%) selective neurons. We see there are many (64 - 32%) inactive neurons (which could become selective for future learning).

More importantly we discover that the model highly overfits to the training examples. This means that in this high dimensionality space, examples of the same object (e.g. the same digit rotated slightly) are easily too orthogonal to be considered the same stimulus by the network. This means the first layer becomes selective to the training examples but does not generalize well to new examples. We will call this the *problem of difference*

In particular, as an example of this we have two examples of the digit 1 in Fig. 3.8. The first one is an example the network has been trained with while the second one is a new example. After the learning process, during retrieval, when the network receives the second one as an input, none of the neurons that fired for the first example, fire (in particular none of the selective neurons). This is due to the fact that the cosine of the angle they form is 0.3919 and thus they are considered as very different stimuli by the network.

3.1.3 Conclusions

We conclude that it is promising that more complex patterns could be used as long as we address *the problem of similarity* (so patterns are sufficiently different to be distinguishable and do not produce conceptual behaviour in the first layer) and *the problem of difference* (so the same patterns showed slightly transformed are recognised as the same).

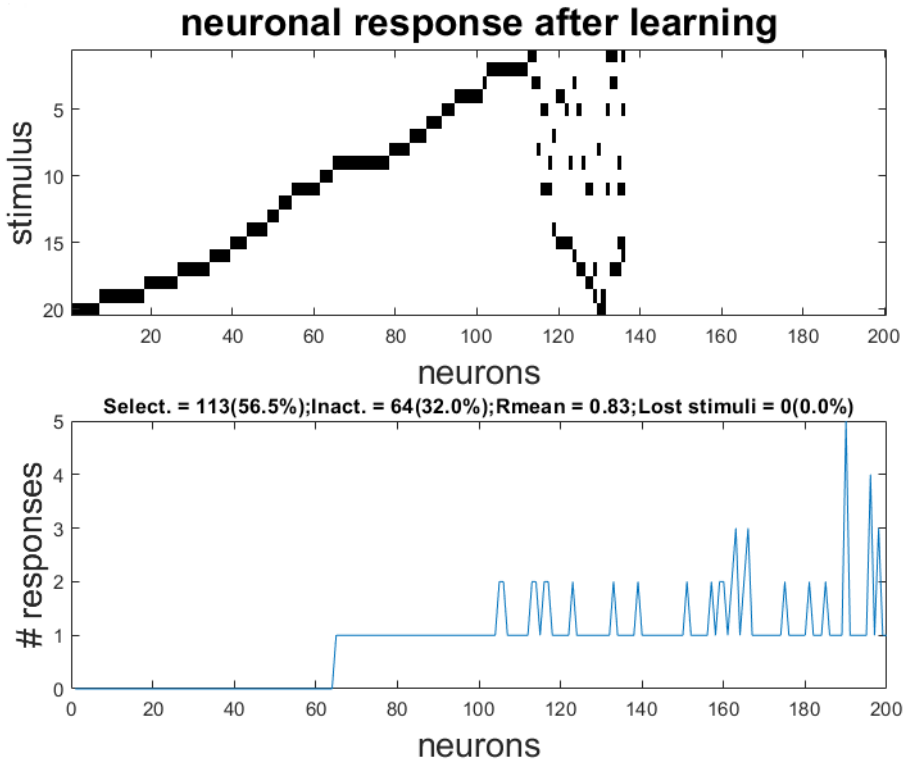

Figure 3.7: Results of the selective layer with digits. Top: Rasterplot showing the activation of neurons to a given stimulus (neurons ordered for visualization) Bottom: Graph that shows the number of responses for each of the neurons (not ordered). We see high selectivity and high inactivity. We measure no lost stimuli.

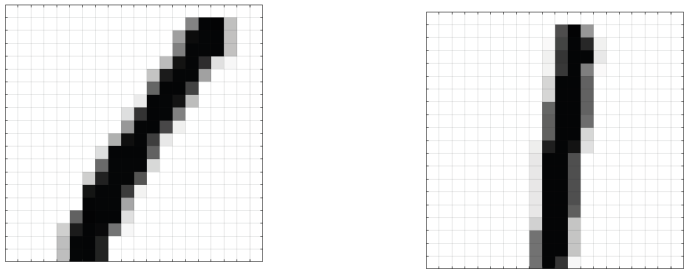

Figure 3.8: Two examples of the digit 1. Even though they represent the same digit, they are considered very different by the network. They are an example of *the problem of difference*.

3.2 Images

We decided to use images of handwritten symbols (numbers, letters, etc.) as patterns to associate. In order to avoid the *the problem of difference*, we do some preprocessing of the images and compute some characteristics of the symbols to ensure that the same symbol represented differently in two images (e.g. rotated) correspond to a very similar stimulus used as input to the network (see Section 3.2.2). To avoid the *the problem of similarity*, we do some study of the symbols we are going to use as well as study the training examples to ensure they are different enough to be distinguished by the first layer (see Section 3.2.3).

3.2.1 Segmentation

Before doing this, we need to segmentate the symbol in the image from the background. We use one of the simplest ways: color. We write the symbols in red with a white background. To segmentate, we then compute the value of redness of each of the pixels with coordinates (x, y) of the image as:

$$Redness(x, y) = R(x, y) - max(G(x, y), B(x, y)) \tag{3.3}$$

where R, G, B are the color components of the image. We then assume a pixel (x, y) is part of the symbol if $Redness(x, y) > thresh$ for some specific threshold we choose statically (in particular we see $thresh = 50$ segmentates correctly the red we use). Furthermore, to smoothen the binary image obtained we use a function to fill up wholes.

3.2.2 Moments

As stated before, in order to avoid *the problem of difference* we have to make sure different images of the same symbol relate to the same stimulus used as input to the network. The transformations a symbol could be affected by are translation, rotation, scaling and perspective. We are going to address the former three by computing 9 invariant moments and sample 400 points of the radius signature. With this, we make the model receive a $409D$ vector which is very similar for any image of the same symbol with regard to these three transformations. In biological systems this corresponds to a previous step of vision processing before associative learning. Perspective will be discussed in depth in Section 3.6.

The computed invariant moments[27] to translation, rotation and scaling are:

- **Eccentricity**: The eccentricity of a symbol is the quotient of the length of the major and minor axis.
- **Compatibility**: The compatibility of a symbol is defined as P^2/A where P is the perimeter and A the area.
- **Hu moment invariants**: Given $f(x, y)$ a function which gives us the intensity at the pixel with coordinates (x, y) of an image, we compute first the *moments* of order

$(p+q)$ for $(p,q) = (0,0),(1,0),(0,1)$ given by:

$$m_{pq} = \sum_x \sum_y x^p y^p f(x,y). \tag{3.4}$$

With them we compute

$$\bar{x} = \frac{m_{10}}{m_{00}}, \qquad \bar{y} = \frac{m_{01}}{m_{00}}, \tag{3.5}$$

used to get the *central moments* of order $(p+q)$ given by:

$$\mu_{pq} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q f(x,y). \tag{3.6}$$

We then proceed to calculate the *normalized central moments* of order $(p+q)$:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \quad \gamma = \frac{p+q}{2} + 1. \tag{3.7}$$

With all of this we can finally compute the 7 Hu moment invariants:

$$\begin{aligned}
\phi_1 =& \eta_{20} + \eta_{02}, \\
\phi_2 =& (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2, \\
\phi_3 =& (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2, \\
\phi_4 =& (\eta_{30} + \eta_{12})^2 + (3\eta_{21} + \eta_{03})^2, \\
\phi_5 =& (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\lfloor (\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2 \rfloor + \\
& (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\lfloor 3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 \rfloor, \\
\phi_6 =& (\eta_{20} - \eta_{02})\lfloor (\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 \rfloor + \\
& 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}), \\
\phi_7 =& (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\lfloor (\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2 \rfloor + \\
& (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\lfloor 3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 \rfloor.
\end{aligned} \tag{3.8}$$

We then normalize them so every moment contributes equally to the representation as follows:

$$\phi'_n = |ln(|\phi_n|)|. \tag{3.9}$$

- **Radius signature**: We now focus on the border of the symbol. The radius signature consists in a function $r(\theta)$ which given an angle θ gives the length of the vector from the centroid of the symbol to the border with angle θ. We use a generalisation for when the centroid lies outside of the border. Given a positive parametrisation of the border as a close curve $\gamma(t) \quad t \in [0,1]$ we compute $r(t^*)$ as the distance between the centroid and the point $\gamma(t^*)$. Computationally, we have a finite list of points of the boundary (x_i, y_i) and we compute the distance to the centroid (c_1, c_2) with the Pythagorean theorem:

$$r_i = \sqrt{(x_i - c_1)^2 + (y_i - c_2)^2} \tag{3.10}$$

We shift it circularly so $r(0) = max(r)$, normalize it $r' = r/max(r)$ and split it in 3 discrete categories as follows:

$$r''(i) = \begin{cases} 1 & r'(i) \geq 0.7 \\ 0 & 0.3 \leq r'(i) < 0.7 \\ -1 & r'(i) < 0.3 \end{cases} \tag{3.11}$$

Even though this reduces greatly the information the signature offers, we need to do it to aid orthogonality of stimuli. In practice, the number of pixels that form the boundary is determined by the size of the symbol in the image. Since we need all the stimuli to be equal in dimension, we sample 400 points equally spaced from $r''(i)$. If the symbol is so far away that it does not have at least 400 pixels of boundary we consider it too far away to be processed.

3.2.3 Data preprocessing

The symbols we have decided to use belong in three categories: numbers, shapes and letters. The model will learn to associate the symbols belonging to the same category together. We will use the term 'concept' to refer to each of these categories. In theory it could learn to associate any subsets of the symbols but for practical usage we will show the symbols belonging to each of the concepts closely in time so the robot makes an association between them and not across concepts. We used the following symbols initially:

- Numbers: 1, 2, 3, 4, 5 and 7.
- Shapes: Triangle, Square, Circle, Semicircle, Star and Bar.
- Letters: G, H, K, L, T and V.

We used these symbols because they have only one border, i.e. they are filled symbols, without holes. This makes the computation of the radius signature easier.

Due to *the problem of similarity* we can not use all these symbols since the moments of many of them are too similar. We decide we are going to study which symbols have moments that are the most different between symbols and the most invariant between examples of the same symbol.

We take 5 examples $(s_i^1, s_i^2, ..., s_i^5)$ of each one of the symbols s_i with more or less same translation, scale and rotation (since they have been checked to be preserved by the moments) but very different perspectives. We then compute, for every symbol, the following:

- An approximation of the false negative error. We compute for every example the number of other examples (of the same symbol) that are different enough to be labelled wrongly as some other symbol and choose the minimum.

$$FN(s_i) = \min_{k=1,...,5} \quad |s_i^j \quad j = 1, ..., 5 \neq k \quad s.t. \quad cos(s_i^k, s_i^j) < 0.5| \tag{3.12}$$

- An approximation of the false positive error. We compute for every example the number of other examples (of different symbols) that are similar enough to be labelled wrongly as the same symbol and choose the minimum.

$$FP(s_i) = \min_{k=1,...,5} \quad |s_l^j \quad l = 1, ..., 18 \neq i \quad j = 1, ..., 5 \quad s.t. \quad cos(s_i^k, s_l^j) > 0.5| \quad (3.13)$$

We plot each symbol s_i as a point $(FN(s_i),\ FP(s_i))$ in the plane. The results are shown in Fig. 3.9. A point more close to the left is a symbol which is easily detected as that symbol independently of perspective. We see that the 'Circle' is easily confused as something different from a circle with perspective. A point more close to the bottom is a symbol which is sufficiently different from others so others are not mixed up with it due to changes in perspective. We see that many symbols are confused easily with the 'Bar' with perspective.

We pick the 3 numbers, 3 shapes and 3 letters such that the sum of these errors is the least and we end up with:

- Numbers: 2, 3 and 7.
- Shapes: Square, Semicircle and Star.
- Letters: G, H and K.

Again, due to *the problem of similarity*, we can not use more than one training example of the same symbol, so we have to select which example of each symbol to use as training. We try to pick the set of examples such that the cosine of the angle between pairs is as small as possible in order to avoid the problem. The optimum set can be computed with a branch and bound algorithm. However, since in general the robot could already have concepts formed when learning new ones, to make it computationally faster and allow on-line learning, we have decided to use a greedy algorithm using the simplification that it is good enough if the cosine of the angles the possibly selected example forms with the already selected ones is less than a certain threshold:

$E \leftarrow [s_1^1]$
for $i = 2, ..., 9$ **do**
  for $j = 1, ..., 5$ **do**
    if $\neg(\max_{ex \in E} cos(s_i^j, ex) > thresh)$ **then**
      $E \leftarrow E \cup s_i^j$
      break
    end if
  end for
end for

In particular, we set a threshold of 0.1 and started increasing it as we saw we could not find one example of each symbol such that they formed a cosine lower than 0.1. We reached

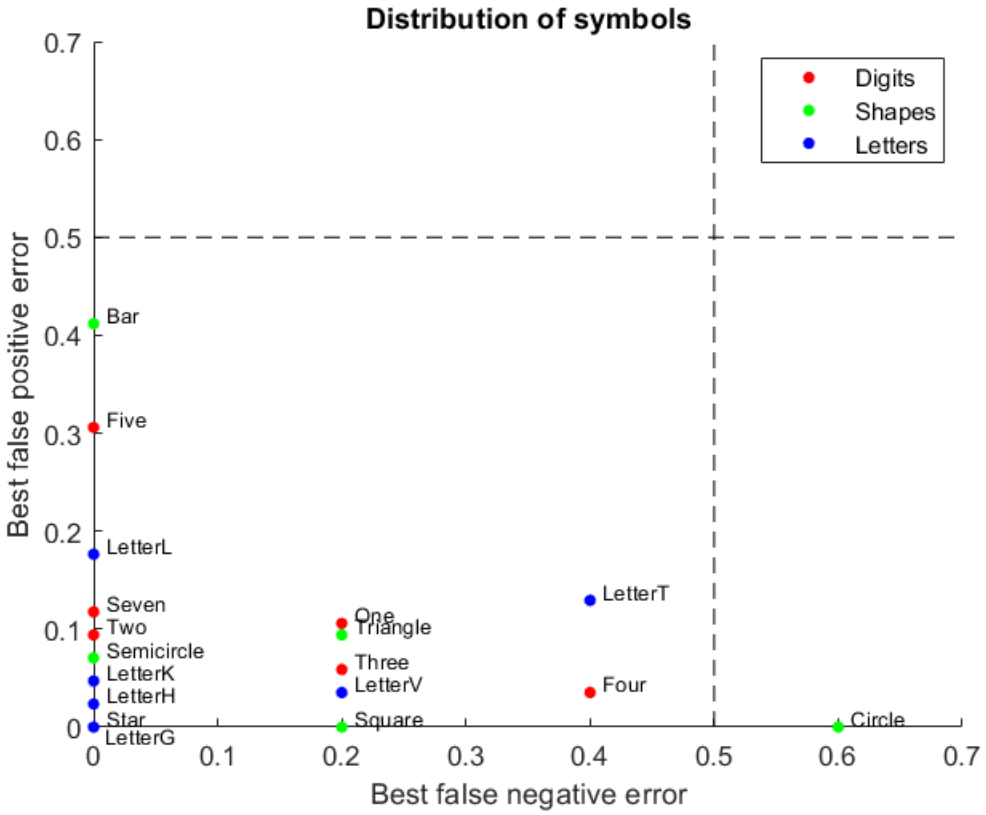

Figure 3.9: Plot of each of the symbols s_i in $(FN(s_i), FP(s_i))$ categorised in digits, shapes and letters. More to the right symbols are more easily not recognisable due to perspective. Other symbols are easily confused with more to the top symbols due to perspective. We pick the symbols such that the sum of these errors is the least.

Figure 3.10: Training examples used for each of the numbers (2, 3 and 7), each of the shapes (Square, Semicircle and Star), and each of the letters (G, H and K).

a threshold of 0.55 which is below the lowest bound that generates *the problem of similarity* and we obtained one example for each of the symbols to use as training set. These can be seen in Fig. 3.10.

3.3 Prediction

Once the network has learnt the system needs to predict to which concept (number, shape or letter) the symbol it has captured the image of belongs to. As stated in Section 2.4 retrieval is done via using as input the computed moments and compute the forward propagation until we end with some activation of the concept layer. However it is in this practical setting that we have to design a way to translate the activation vector $\boldsymbol{y}_{cn}$ to a specific prediction (either 1, 2 or 3).

In order to do that we need to compute an extra structure called 'the concept map'. The concept map $(dict_i)_{i=1}^N$ is an array with as many components as neurons there are in the concept layer that has the value 1, 2 or 3 in its position i if the i-th neuron is conceptual representing the numbers, shapes or letters, respectively; or a value of -1 if it is not conceptual. We consider a neuron to be conceptual if it fires to the training examples of all the symbols of a concept and only to them.

Then, when we have the activation $\boldsymbol{y}_{cn}$ of the concept layer for a new example, we compute the sum of the firings of the conceptual neurons of each of the concepts and choose the one with maximum sum as prediction.

$$pred = argmax_{i=1,2,3} \sum \boldsymbol{y}_{cn}[dict = i] \tag{3.14}$$

If no concept neurons fire then $pred = -1$.

Initially we added a factor of certainty in order to know how sure the network was of its prediction, a very valuable measure to have. It was computed as the percentage of activation

the concept cells of the predicted concept over the activation of all the concept cells.

$$cert = \frac{\sum \boldsymbol{y}_{cn}[dict = pred]}{\sum_{i=1,2,3} \sum \boldsymbol{y}_{cn}[dict = i]} \tag{3.15}$$

If no concept neurons fire then $cert = 0$.

However, in Section 3.5, we present some findings certainty as a measure gave us and justify why we do not use it for our final version of the model.

3.4 Parameter tuning

During the testing of the model we encounter that the values recommended by the paper of θ_{cn} and β_{cn} do not properly generate concept cells in the concept layer through learning with these complex stimuli. Thus, we decided to explore different values of θ_{cn} and β_{cn} and evaluate how good the concept layer learns with each of them. In particular we use a score to qualitatively evaluate if the layer has correctly learned a predetermined number n of concepts as follows:

$$ConScore_n = \begin{cases} m \quad m = 0, 1, ..., n \quad \text{if } m \text{ concepts have been learned.} \\ n+1 \quad \text{if } n \text{ concepts have been learned and} \\ \qquad\qquad \text{there are enough non-conceptual neurons to learn a new concept.} \end{cases} \tag{3.16}$$

where we consider that a concept has been learned if there are at least 5% of concept neurons representing that concept. On the other hand, we consider there are enough non-conceptual neurons if there are at least also 5% that do not fall into the past classification. The 5% threshold is to ensure this is not happening by randomness but in general more conceptual neurons of a given concept does not mean better results.

Since this is a very blunt measure we also used the accuracy of prediction over a validation set of examples, formed by the examples used during the data processing phase that were finally not used as training examples.

We therefore compute $ConScore_3$ and the precision of the concept layer with $\beta_{cn}, \theta_{cn} \in \{0, 0.125, 0.25, 0.375, 0.5\}$ and choose the one which has the greatest $ConScore_3$ and, in case of draw, the one with greatest precision. We order the preferences in this way since it matters more that the system learned all the concepts even if it has overall less precision rather than it learned one or two concepts but learned them with high fidelity. However, we expect that in general both measures are highly correlated.

The results obtained are shown in Table 3.1. We choose the set of optimal parameters $(\beta_{cn}, \theta_{cn}) = (0.25, 0.125)$.

3.5 Misclassification and ignorance

Another parameter we initially wanted to tune its certainty. As stated in Section 3.3, the model will output a measure of how certain it is in its prediction. We expected that the error

Parameter tunning: $ConScore_3$/precision					
$\beta_{cn} \backslash \theta_{cn}$	0	0.125	0.25	0.375	0.5
0	0/0.222	0/0	0/0	0/0	0/0
0.125	0/0	4/0.7222	4/0.4722	4/0.25	2/0.0278
0.25	0/0	4/0.7778	4/0.6667	4/0.3889	4/0.3056
0.375	0/0	4/0.75	2/0.7222	4/0.5833	4/0.3611
0.5	0/0	2/0.25	2/0.7222	2/0.6667	2/0.5

Table 3.1: $ConScore_3$ and precision computed for each value of β_{cn} and θ_{cn}. In grey cells with optimal $ConScore_3$ (value 4) and in a darker grey the one with also optimal precision.

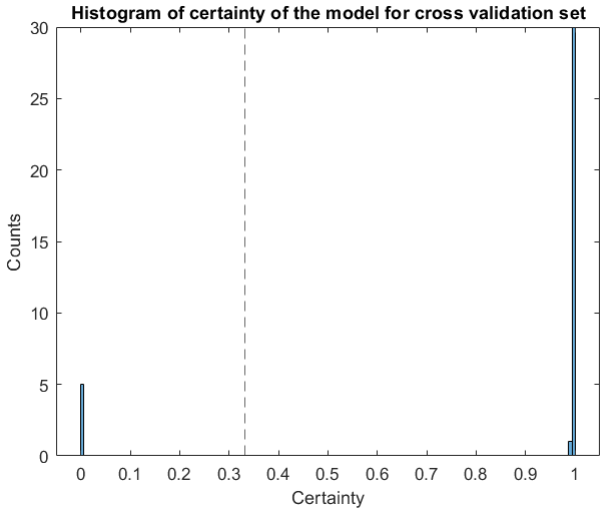

Figure 3.11: Histogram of certainty computed for the validation set. We see a very binary behaviour since all the values different from 0 (with possible domain $[0.\overline{3}, 1]$) are accumulated next to 1.

and the uncertainty of the model would be related and we could find a threshold such that if the classifications with certainty below that threshold were discarded, we would achieve better precision. However, on the validation set, as shown in Fig. 3.11, we encounter a very binary behaviour: either almost only one type of concept neuron fires ($cert \approx 1$, $pred \neq -1$), or no concept neurons fire ($cert = 0$, $pred = -1$). In particular the cases with certainty a little less than 1 are correctly classified. Therefore it is redundant information with the value of $pred$. We decide to drop the concept of certainty and substitute it for equivalently splitting the error into misclassification (some concept neurons fired but they where not the correct ones, $pred \neq -1$ but has an incorrect value) and ignorance (no concept neurons fire, the system believes it has never seen the symbol before, $pred = -1$).

In practice misclassified examples cannot be noticed (until already facing the consequences of the misclassification) but we can profit from the ability of our system to show itself ignorant to fix that type of error. In particular, the most simple solution is to force the system to move and try to resample the stimulus in a different manner until it stops being ignorant. This is similar of how some biological systems tend to move their position to see the object differently when they cannot easily recognise it. The particular way in which it has to do it and a justification of why we expect the system to stop being ignorant in this way will be seen in Section 3.6.

If we get rid of the ignorance error through resampling and we only take into account misclassified examples we get an increase of the accuracy on the validation set from $0.\overline{7}$ up to 0.903226. (Misclasification is $\approx 44\%$ of the error while ignorance is $\approx 56\%$).

3.6 Perspective

In this section we will discuss the importance of perspective in error. Specifically, in the ignorance error.

As stated in the Section 3.2.2 the main ways in which a symbol can be captured transformed are translation, rotation, scale and perspective. Using invariant moments to the former three makes perspective the main factor in error.

To support this claim, we manually label the validation examples as with very different perspective or similar perspective to the training example of the same symbol. Now we find relations of conditional probabilities and implications in our validation set which will relate to high correlation in the population.

Mathematically, if R is 'the model correctly classified' and P is 'the example is in different perspective' we observe that, in our sample, $\mathcal{P}(P|\neg R) = 1$ (all of the wrongly classified examples are different in perspective) or in this case equivalently, $\mathcal{P}(R|\neg P) = 1$ (all of the examples similar in perspective where correctly classified). We conclude error is directly related to bad perspective ($\neg R \Rightarrow P$) and (by contrapositive) good perspective to correct classification ($\neg P \Rightarrow R$), respectively. In this sense the two are related. Notice that bad perspective is not directly related to error ($P \not\Rightarrow \neg R$) and (by contrapositive) correct classification is not to good perspective ($R \not\Rightarrow \neg P$) since there are examples that where considered both in bad perspective and correctly classified.

Now we can discuss how resampling to solve ignorance errors should be done and why we expect it to work (see Section 3.5). We see that $\mathcal{P}(\neg I|\neg P) = 1$ (all of the examples similar in perspective do not make the model ignorant) and thus good perspective is directly related to no ignorance ($\neg P \Rightarrow \neg I$). Therefore, if the movement of the system for resampling is done in such a way that we change perspective of the image over and over, we expect at some point the system will sample in a similar perspective to that of the training example, stop being ignorant (finally realise what symbol is) and probably classify correctly the symbol (remind $\mathcal{P}(R|\neg Ignorant) = 0.903226$).

All of this discussion will be solved once we upgrade the preprocessing unit with feature extraction (invariant to perspective) in Section 3.8.

3.7 Results

We used an architecture of $409D$ stimuli (9 moments and 400 sampling points of the radius signature taken from the training set concluded during data preprocessing (see Section 3.2.3)), $M = 300$ neurons in the first layer and $N = 150$ neurons in the second. We show each stimuli for $t_\sigma = 1$ cyclically up to $T = 300$ with an $\alpha = 30$.

First we analyse the results of the first layer with parameters $\beta = 0.7$ and $\theta = 0.8$ shown in Fig. 3.12. At the top we see the values of the $409D$ stimuli with one color for each stimulus. Then we see the activation rasterplots and the number of responses per stimulus before learning and after learning. As we can see, we achieve high selectivity: we increase from 79 (36.3%) to 210 (70%) selective neurons. However we do not change the number of inactive neurons: 70 (23.3%) in both cases. We also see inhibition ensured a more equitable share of the stimuli between selective neurons (in the rasterplot in c)) and thus there are no lost stimuli.

The results of the second layer with the optimal parameters, computed in Section 3.4, $\beta_{cn} = 0.25$ and $\theta_{cn} = 0.125$ are shown in Fig. 3.13. As before, at the top we see the values of the $M = 300D$ activation vectors of the previous layer for each stimulus. Now, instead of showing the number of responses per neuron we show in a histogram how many neurons are nonconceptual (class -1) and how many conceptual to numbers (class 1), shapes (class 2) or letters (class 3). We observe that the layer changes from having almost all (149, $99,3\%$) nonconceptual neurons to becoming highly conceptual for the three concepts. In particular we achieve a $ConScore_3 = 4$ (maximum score) and the following shares: 33 (22%) conceptual neurons for numbers, 28 (18.7%) for shapes, 39 (26%) for letters, and 50 (33.3%) nonconceptual neurons for future concept learning.

We use a set of test examples not used in the previous sections for testing the precision of the model to correctly predict the class they belong to: number, letter or shape. They include pictures with very different perspectives as well as slightly translation, rotation and scaling. We accomplish an accuracy of $0.\overline{7}$. However, if we consider only the misclassification error (wrong concept neurons fire) and not the ignorance error (no concept neurons fire) we achieve a precision of $0.97\overline{2}$.

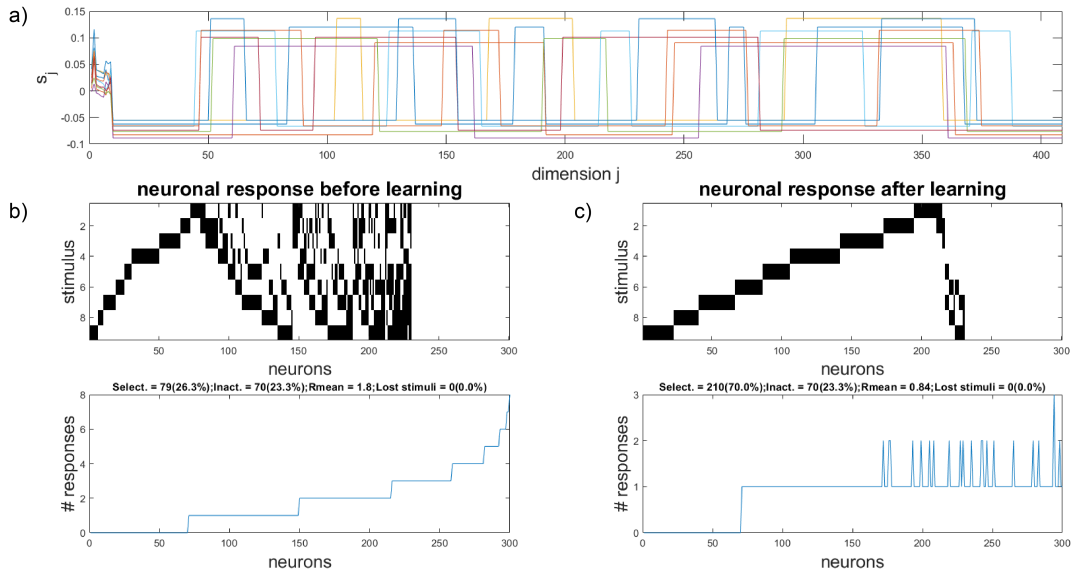

Figure 3.12: Results of the selective layer. a) In different colours, stimuli used. 9 moments and 400 points of the radius signature. b) Results of randomly initialised weights. c) Results after training. Much higher selectivity and no lost stimuli.

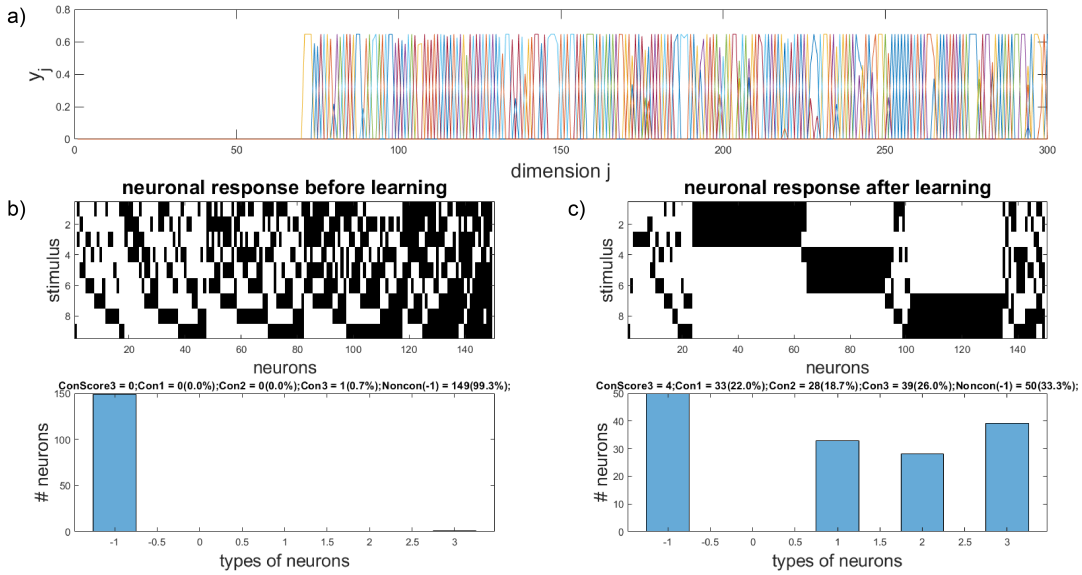

Figure 3.13: Results of the concept layer. a) In different colours, activation of previous layer for each stimuli b) Results of randomly initialised weights. Almost all neurons are nonconceptual c) Results after training. High number of conceptual neurons for each of the concepts.

3.8 CNN

As we have justified in Section 3.6, the error due to ignorance, that is, no conceptual neurons fire, is due to the fact that the representation computed for a given image via its moments (Section 3.2.2) is not invariant to perspective. Therefore, examples of the same image with different perspectives are not considered as the same symbols by the system.

In order to solve this, we have decided to try changing the representation of a given image by its moments for a representation based on some features of the symbol represented. This way we intend to reduce the problem of ignorance significantly since the new representation used would be invariant to perspective. To compute these features we focus towards feature extraction with convolutional neural networks (CNNs).

A CNN is a deep neural network based on convolutional and pooling layers. They are widely used for computer vision tasks. In particular we use a residual neural network (ResNet) which is a kind of ANN where there are 'skip' connections between layers to avoid a 'degradation' problem and therefore facilitate learning[28].

Since training of CNNs needs a high amount of training data and computation, we have decided to use an already trained CNN for image classification for feature extraction. The goal is to use the activation of one of the layers of the network on the input of the image, as stimulus for the hippocampus model.

We use ResNet-50[29], a residual CNN with 50 layers trained for the task of image classification for the ImageNet database[30]. ResNet-50 is available in the Deep Learning Toolbox of MATLAB.

Since the input to ResNet-50 required is a 224×224 RGB image, we use the color segmentation to localize the symbol and then crop and rescale the original RGB image in a square of 224×224 pixels that bounds the symbol.

To choose which layer to use we compute for the validation set the approximation of the false negatives and false positives the representation of a given layer of the network would give us in a similar way as in Section 3.2.3. Since we want to solve the problem of ignorance, that is, the fact that examples of the same symbol are not classified as such, we give more weight to the false negatives (80%) than the false positives (20%) and choose the layer that minimises this weighted sum.

We conclude that best layer to use for feature extraction as stimuli is one of the last ones, in particular the layer $activation\_48\_relu$.

In Fig. 3.14 we see color coded the cosine of the angle each of the representations of the examples of the validation set form with each other. In (a) we see the angles from the representations of the image moments. Some examples show high orthogonality. For example: examples 25 to 28, all belonging to the letter G, are easily identified as examples of the same symbol and distinct from the rest, as shown by the red box in a mostly green and blue row/column. However, we also see examples mix. For example, examples 29 to 36 belong both to the letter K and the letter H but are indistinguishable between each other. In (b), the angles of the representation by an intermediate layer of ResNet-50 ($activation\_38\_relu$) are shown. We see all the vectors are very similar. In (c) we have the representation of the layer $activation\_48\_relu$. In particular, we observe squares that represent that examples of

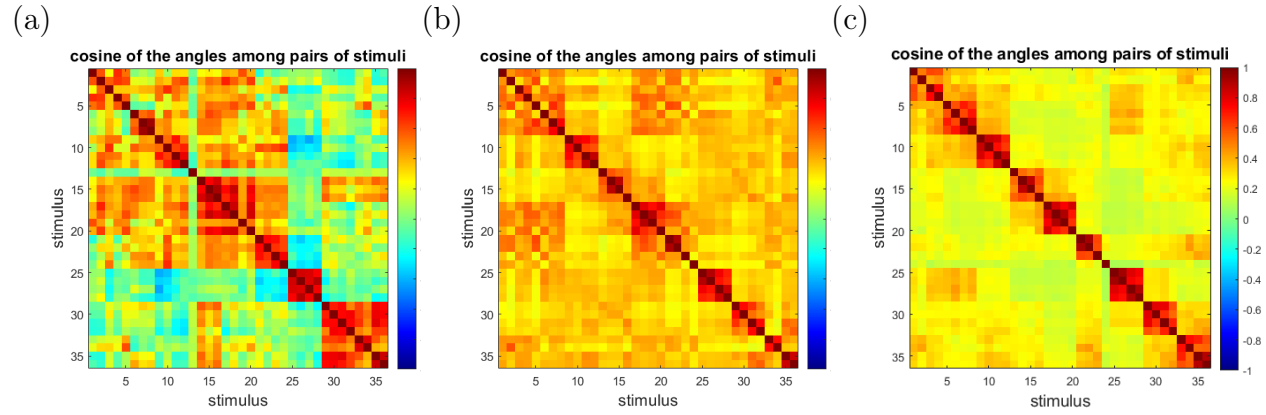

Figure 3.14: Cosine of the angle between the representation of the different stimuli in the validation set. (a) Image moments representation. Some symbols are very distinct but others are easily mixed. (b) Activation of *activation_38_relu* representation. All symbols are mixed. (c) Activation of *activation_48_relu* representation. Examples of the same symbol are similar and those of different symbols are distinct enough.

the same symbol form small angles (red) while examples of different symbols form a bigger angle (orange or green).

3.9 Results CNN

We use the same parameters as before except that we decrease the value of the threshold $\theta = 0.6$ (to help stimuli slightly misaligned with the weights vector also produce activation) and we use as stimuli the vectorized activation of the layer *activation_48_relu* of the ResNet-50 with 25088 dimensions instead of the image moments.

In Fig. 3.15 we see the results of the selective layer. We see very similar results as with the moments preprocessing. A total of 214 (71.3%) of selective neurons with no lost stimuli and 60 (20%) inactive neurons.

However, we see different results in the concept layer in Fig. 3.16. We observe that after learning we get a smaller number of neurons conceptual to each of the concepts: 17 (11.3%) conceptual neurons for numbers, 13 (8.7%) for shapes, and 11 (7.3%) for letters. However since all of them are above the 5% minimum required we consider these results as having a $ConScore_3 = 4$ (maximum). We also see that in the greater number of nonconceptual neurons (109 or 72.7%) we have more inactive neurons than in the case of the image moments.

Neither the fact that we have less conceptual neurons for each concept nor that more nonconceptual neurons are inactive means we should consider these as worse results. In fact, we obtain an accuracy of 1 on the test set (taking into account both ignorance and misclassification errors) which is a huge improvement of our previous results with the image moments and an optimum precision overall.

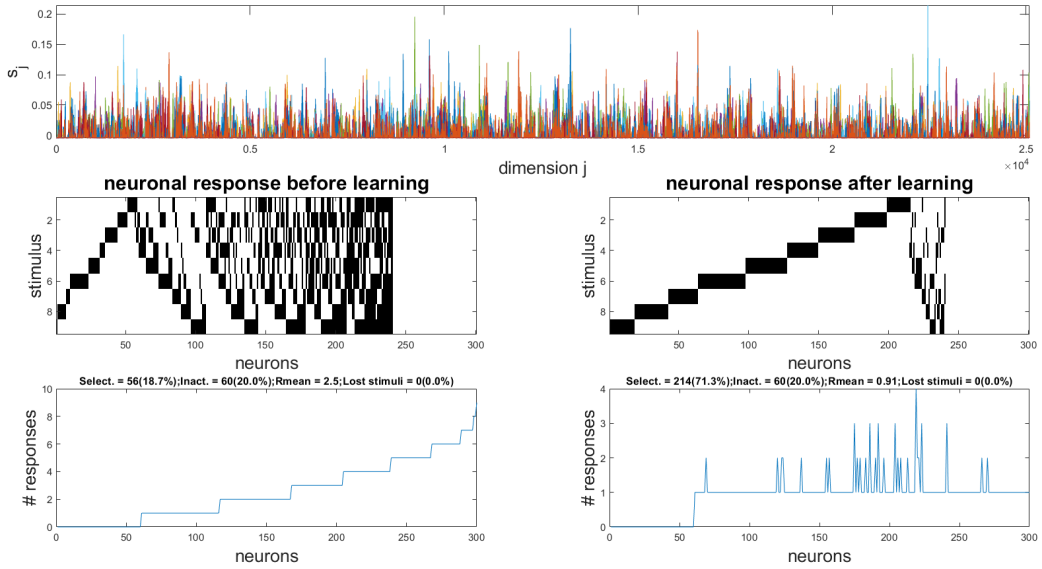

Figure 3.15: Results of the selective layer with CNN. a) In different colours, the 25088D stimuli used. b) Results of randomly initialised weights. c) Results after training. Very similar results to those obtained with image moments.

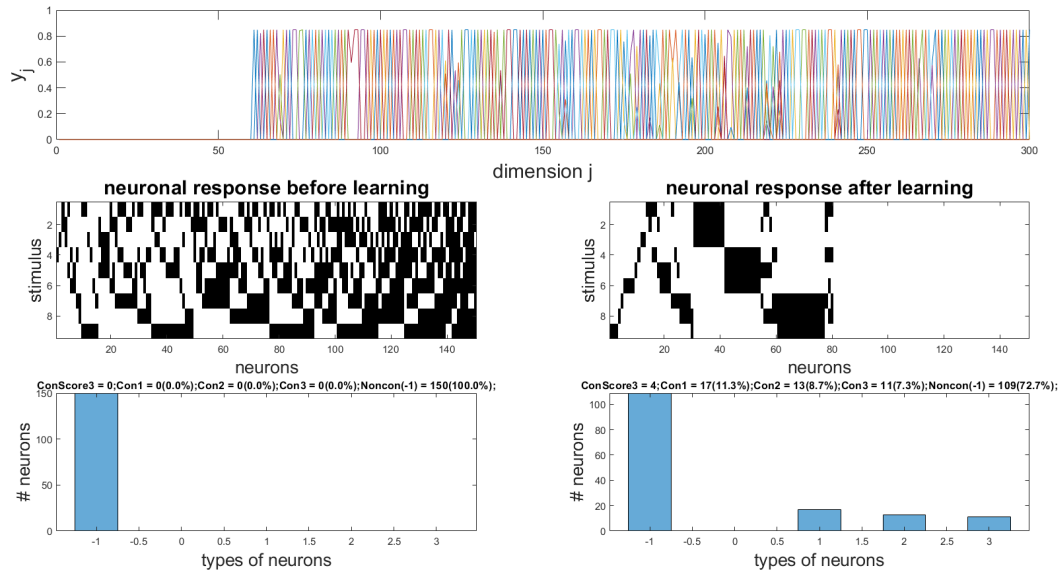

Figure 3.16: Results of the concept layer with CNN. a) In different colours, activation of previous layer for each stimuli b) Results of randomly initialised weights. All neurons are nonconceptual c) Results after training. Fewer conceptual neurons for each concept than with image moments but still above the minimum 5% required.

Chapter 4

A case of use. Episodic memory in a mobile robot

4.1 Robot

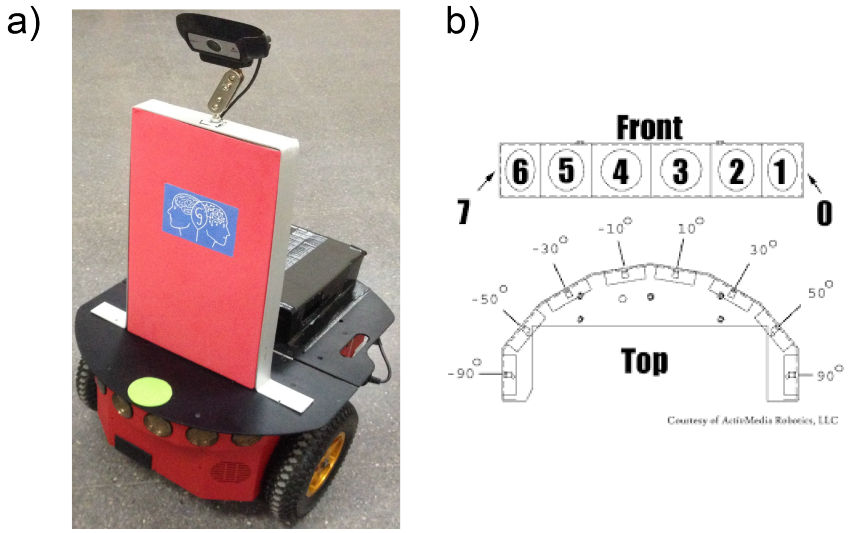

Figure 4.1: a) Pioneer robot with camera installed. b) Diagram of the distribution of front sonars[31].

The robot owned by the Cognitive Systems and Neurorobotics research group is the Pioneer 3DX, a small mobile robot used for research purposes (Fig. 4.1a)). It consists of a main body with three wheels: two on the side and one in the rear. It is a differential wheeled robot, i.e. the ones on the side have an independent motor each while the third is for support.

The two main ways the robot captures information is through a webcam that has been installed on top of the base at $\approx 70cm$ height and the sonars. The camera is connected to the PC on board an can capture video and image. There are 8 sonars distributed in the

front as shown in Fig. 4.1b). They can detect when an object is near and the number and distribution allows for nuanced sensing.

Finally, and even though the robot does not have it yet, we have considered the possibility of installing and using a gripper that can close and open. We use the specifications of the Pioneer 2 gripper[32] (adaptable to the Pioneer 3DX) but in general, any gripper would work.

The robot brings ARIA, an interface to send and receive signals from the robot using C++. In particular, the research group uses the PC on board to run MATLAB scripts that execute MEX functions in C++ which connect with ARIA.

4.2 Experiment design

The sonars will give us detailed information of the environment. This, joined to the high mobility two independently motored wheels give the robot, will allow it to navigate spaces easily. Since we want it to have a predictable behaviour we decided to use corridors to indicate the path the robot has to follow and use the sonars (besides keeping straight in the corridor) for turning and for solving the tasks. In particular, since we use the sonars for navigation, the camera can focus on taking samples of the symbols to test the associative memory model.

Therefore the skeleton of our experiments would consist on the robot exploring a space determined by corridors until some specific circumstances occur. In that moment, the robot would take an image of a symbol situated on a wall and depending on the concept the symbol belongs to, the robot will have to perform one or other action. These actions can involve either moving (using the wheels) or grabbing/releasing something (using the gripper) or a combination of the two. After finishing the action the robot needs to be again in a corridor-like space to continue moving and start the cycle again.

The two experiments we have decided to test using this scheme are: oriented navigation and order execution.

- In **oriented navigation** we will test the ability of the robot to go from a starting point to a finishing point in a maze. In order to know in the bifurcations whether it has to turn left, right or stop, we use the concepts previously acquired through offline learning: number, shape and letter. When reaching a bifurcation the robot will find a symbol on the wall in front. If this symbol is a number, it would mean the robot needs to turn right; if it's a shape it needs to stop (since it has reached the destination point) and if it's a letter, the robot needs to turn left.

- In **order execution** the robot will move cyclically between three stations. In each one of them it will find a symbol on the wall which will represent one of three different tasks: let go of a box it already has picked up (represented with a number), activate a machine (represented with a shape), or pick up a box (represented with a letter). Once it has performed the task it will move on to the next station.

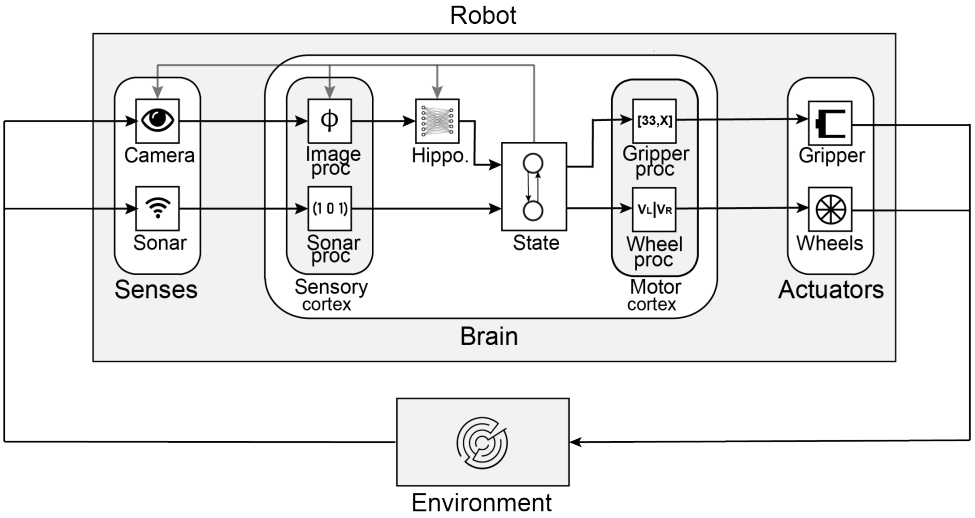

Figure 4.2: Diagram of the conceptual modules used. In the simulation, the environment, senses and actuators are fully simulated.

4.3 Algorithm design

We designed the algorithm in modules that resemble that of the anatomical and functional structures of biological systems.

In Fig. 4.2 we see the structure of modules used. When designing the simulation we distinguish two main parts: the environment and the robot.

- The **enviornment** module keeps track of the state of the environment. It modifies the state, generates the visualization and communicates the position of the robot to the simulated senses. This module will not be used in the real life application.
- The **robot** module further subdivides into three submodules:
  - The **senses** module corresponds with devices that get information from the external environment. In the case of our experiment we use the *sonar* sensors of the Pioneer 3DX to help navigate the space without colliding and a *camera* attached to it to get the images of the symbols later used for association. We had to simulate the behaviour of the real camera and sonars in the simulation. These modules will be changed to the modules of the actual camera and sonars in the real life application.
  - The **brain** module corresponds with all the logic of the algorithm. It consists of two modules (*sensory cortex*) for analysing the data received from the senses module, the *hippocampus* module for stimuli association, the *state* module for decision making and another two modules (*motor cortex*) for transforming the state to actual actions for the actuators modules. This module will be completely unchanged in the real life application.

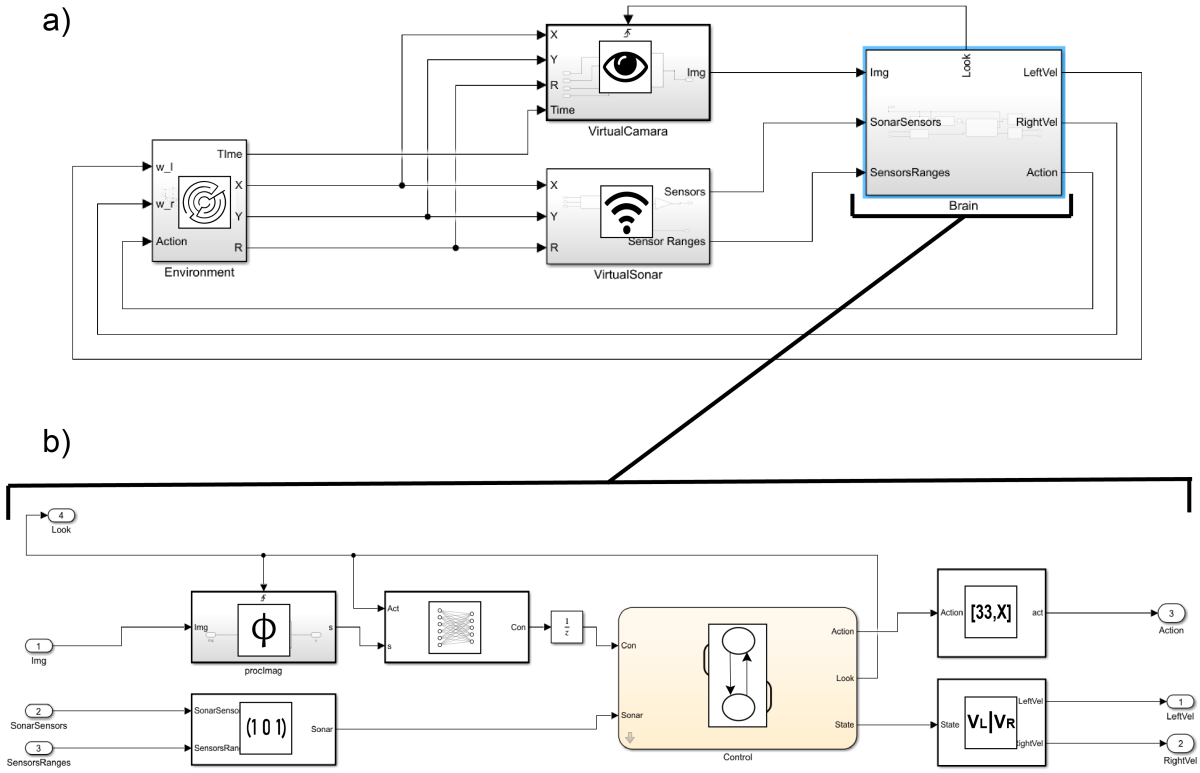

Figure 4.3: Simulink model for the simulated experiment. The modules are directly related to those in Fig. 4.2. a) General modules. Here there are no virtual actuators because they are integrated in Environment. b) The modules that constitute the brain.

– The **actuators** module corresponds with the devices that output the actions to the external environment. In our case we use the *wheels* of the robot and a installable *gripper*. In the simulation we had to simulate the interaction of these modules with the environment. These modules will be changed to the modules of the actual wheels and gripper in the real life application.

In Fig. 4.3 we see the correspondence to the actual Simulink model in the case of the simulation. The modules are directly related to those in Fig. 4.2, except that the actuators are integrated inside the environment module.

4.3.1 Environment

This module consists mainly of the invariant arena and the mutable robot, machines and boxes. The states of the changing parts consist of:

- Robot: the location of the robot and the rotation matrix characterizing the angle with respect to the initial position.
- Machines: a binary value whether the machines are activated or not.
- Boxes: four binary values per object which determine if the object is in station 1, station 2, station 3 or held by the robot.

The environment module completes three main tasks:

- Receive as input the actions of the robot through the actuators (wheels and gripper) and change concordantly the state of the environment.
- Generate the visualization of the environment.
- Give as output the state of the robot to the virtual camera and virtual sonars.

4.3.2 Virtual senses and actuators

- Virtual senses

Camera:

We simulate the camera system using the location of the robot given by the environment as (x, y) coordinates and the rotation matrix $rotMat$ which contains the information of how the robot is oriented. If the input signal from the brain 'Look' is activated, we check whether the robot is in an area we consider the vicinity of one of the symbols and whether it is looking in its direction. To do this we compute the angle θ of the robot using the following definition:

$$rotMat = \begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix} \tag{4.1}$$

If the robot is roughly looking in the direction of the symbol, the camera will load an image with it. This is done by checking whether the angle θ falls in the correct sector of the unit circle (N, E, W and S) as shown in Fig. 4.4. Furthermore, depending on the angle it is looking at, we load the image that resembles most in perspective to what the camera would capture in that situation. This corresponds to the subsector it belongs to (NEE, NE, N, ...).

We output a blank image otherwise. Even though in reality it would capture something different from a blank image, e.g. a corner between two walls, since segmentation is done over the color red it would be equivalent to capturing a blank image.

Sonars:

In order to simulate the sonar system we computed a binary matrix that represents the floor plan of the arena. It has a 1 where a wall or machine is located and a 0 otherwise (boxes are not high enough to be detected by the sonars). In each timestep we use the specifications of the Pioneer 3DX sonars (see Fig. 4.1b)) to compute a segment of line that

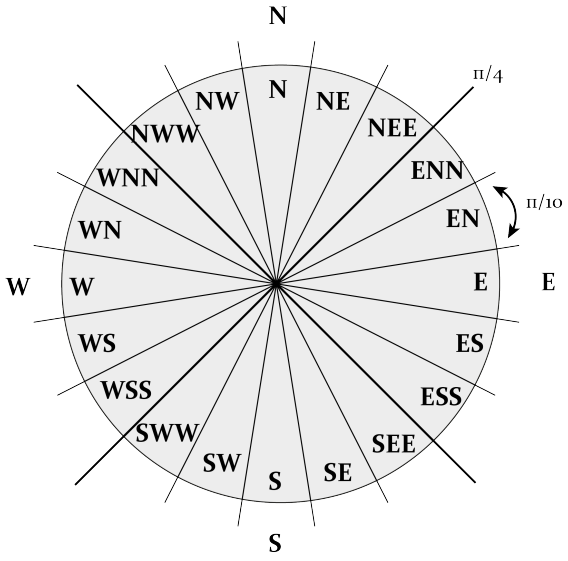

Figure 4.4: Sectors of the virtual camera. If the robot is looking in the correct sector it will get an image of the symbol it is looking to. Depending on which of the 5 possible subsectors the angle falls into, the image loaded will be with more or less perspective correspondingly.

connects the centre of the semicircle the sensors form and the furthest point the sensors can reach. Then we use the binary matrix as a look up table. If any of the segment points has a 1 in the table it means it has reached a wall and therefore the sonar senses it. To transform this so the sonar processing unit can receive it as input we compute the *OR* of all the values of the table in the segment and multiply it by the maximum range of the sonar. This way, if the sonar detects an object the value sent as output will be the maximum range of the sonar and if it does not it will be a 0.

- Virtual actuators

Gripper and Wheels:

In the case of the simulation, there are no such modules but rather they are integrated directly in the part of the environment module that computes the modification of the environment state. In the case of the real life application these modules correspond to the actual gripper and wheels of the robot that receive input from the corresponding processing modules of the motor cortex.

4.3.3 Brain

This is the module with the associative memory model and the logic of the algorithm. It processes the information and outputs an action correspondingly. We can see its modules components in Fig. 4.3b) and we are going to explain them in detail here:

- Sensory cortex

Image processing:
This module receives the image from the camera module. It only activates when it receives the 'Look' signal from the state module, crops and rescales the image into a 224×224 where the symbol is, and computes the activation of the layer *activation_48_relu* of the ResNet-50 to the transformed image.
Sonar processing
The input from the sonar module comes as a value between 0 and a maximum value, the range of the sonar. In this module we check whether the value is strictly less than the range in which case we conclude the sonar is detecting a wall.

- Hippocampus
Here we include the associative memory model. It receives the representation computed by the image processing module and, using the weights obtained during learning ($\boldsymbol{W}$ and $\boldsymbol{U}$), the thresholds (θ and θ_{cn}) and the concept map ($(dict_i)_{i=1}^N$) of the neural network, it predicts the concept associated to the symbol captured by the camera as described in the Section 3.3. The value of the concept is passed to the state module which takes this information and makes a decision based upon it.

- State module
The state module is a stateflow system which changes upon the inputs from the hippocampus and the sonar processing module. This is the only part that differs in the navigation and in the order execution tasks. Mainly it consists of a main state ('going straight') and it can change to states to adjust in the corridor to avoid crashing into walls and to make the correct turns (in the navigation task turning at the corners and in the order execution tasks turning in the middle bifurcation to iteratively go to each station).

Under specific circumstances the robot will enter the 'Look' state. Here, it stops moving and activates the camera module, the image processing module and the hippocampus and waits for the hippocampus to send the prediction of to which concept the symbol captured belongs to. With this information, it performs one of three possible actions or enters a state of resample. The latter is a state used when the system shows itself ignorant of the symbol and wants to retake a picture. The robot backtracks a little, turns a little bit in place, and goes to the 'go straight' state again, hopefully approaching again the symbol but form a slight different angle.

In the navigation task, when all the front sonars are detecting a wall but the left and right sonars are not, the robot concludes it is in a bifurcation and enters the 'Look' state. Analogously, in the order execution task it enters the 'Look' state when the left sonar is detecting the machine, the front sonar is detecting the wall but the right sonar is not detecting anything.

For more details into the specific stateflow diagrams, see Appendix A.

- Motor cortex
Wheels processing:
This module gets as input the state from the state module and outputs the angular

Wheels processing	
State	(v_L, v_R)
Go straight	(v_{front}, v_{front})
Stop	$(0, 0)$
Go back	$(-v_{back}, -v_{back})$
Turn left in place	$(v_{turn}, -v_{turn})$
Turn left tight	$(v_{turn}, v_{turn}/5)$
Turn left wide	$(v_{turn}, v_{turn}/2)$
Turn right in place	$(-v_{turn}, v_{turn})$
Turn right tight	$(v_{turn}/5, v_{turn})$
Turn right wide	$(v_{turn}/2, v_{turn})$

Table 4.1: Relation between states and the wheels velocities (v_L, v_R).

Gripper processing		
State	(P2OS command,argument)	Action
Release object	$(33, 1)$	GRIPopen
Grab object	$(33, 2)$	GRIPclose
Activate machine	$(33, 2)$	GRIPclose

Table 4.2: Relation between states and the gripper commands.

velocity of the left and right wheels, v_L and v_R respectively. Using specified values for v_{front}, v_{back} and v_{turn}, we make the conversion shown in Table 4.1.

Gripper processing:

We used the specifications of the Pioneer 2 Gripper[32] to send the instructions as a vector with two components: the P2OS command number and the argument of the command. Correspondingly we use the conversion shown in Table 4.2.

We consider for simplification that the switch to turn on the machines has some kind of mechanism where the gripper closing activates it, e.g. pressure sensor.

4.4 Results of the simulation

In Fig. 4.5 we see a diagram explaining the correct execution of the oriented navigation task. Likewise, in Fig. 4.6 the same is done for the order execution task. These diagrams are made from stills taken from a video available in the repository.

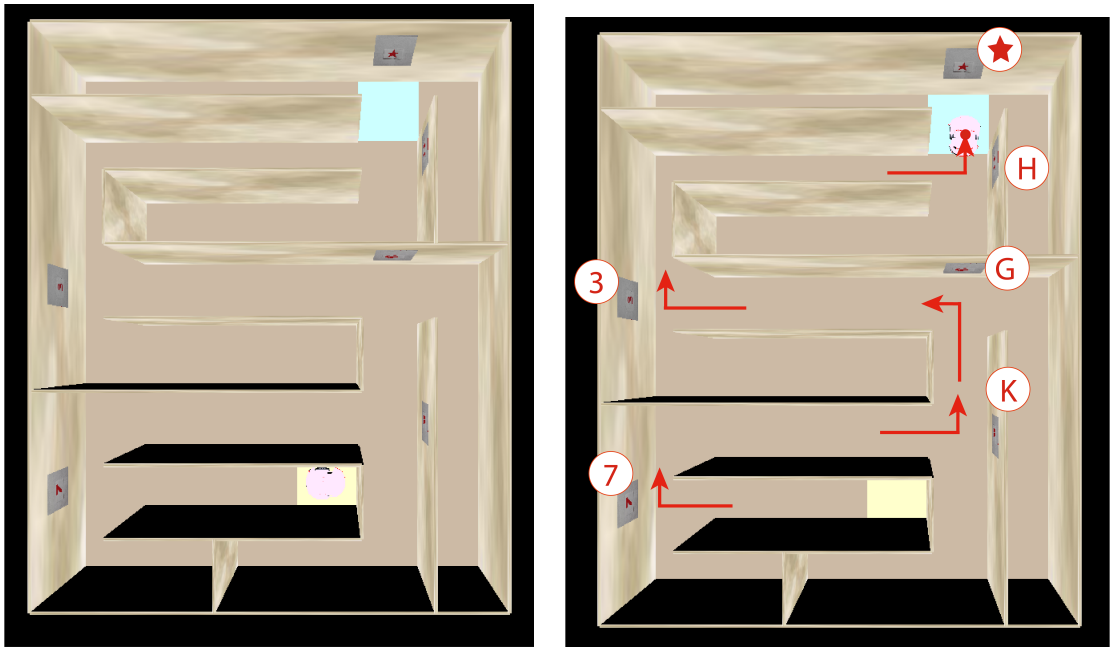

Figure 4.5: Oriented navigation. On the left we have the initial state: the robot starts in the initial base (yellow) and has to reach a goal base (blue) and stop on top of it. On the right we see the correct execution of the task: if the robot detects a number it turns right, a letter, it turns left and a shape, it stops.

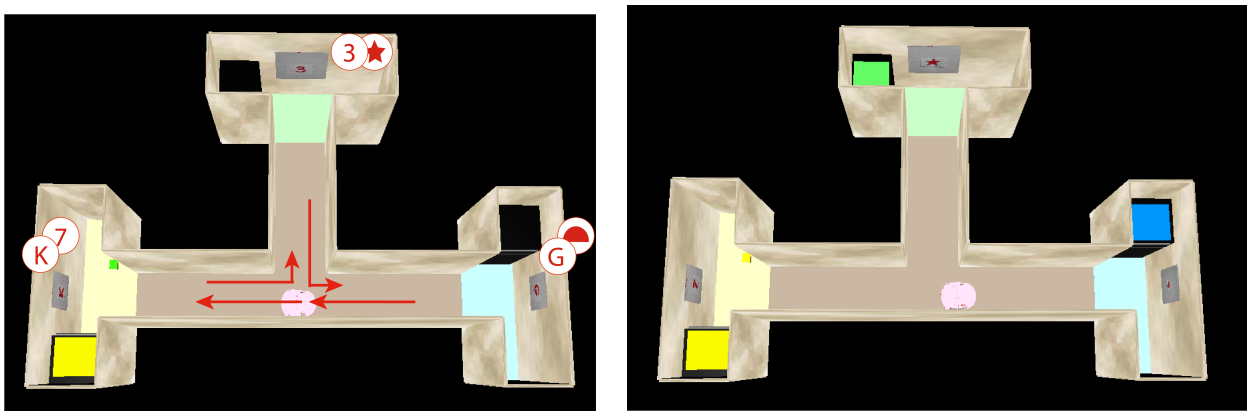

Figure 4.6: Order execution. On the left we have the initial state. The robot moves from station to station cyclically (yellow, green and blue). At each station it encounters an indicator that tells it what action to perform. It visits each station twice (in each station we can see the symbols shown each time). In particular, if it detects a letter it picks up the box, if it detects a number it releases a box it is already holding and if it detects a shape it activates the machine in the station. On the right the final state result of the correct execution of the task: the green box (initially in the yellow station) is in the green station, the yellow box (initially in the blue station) is in the yellow station, and the blue and green machines (initially off) are on.

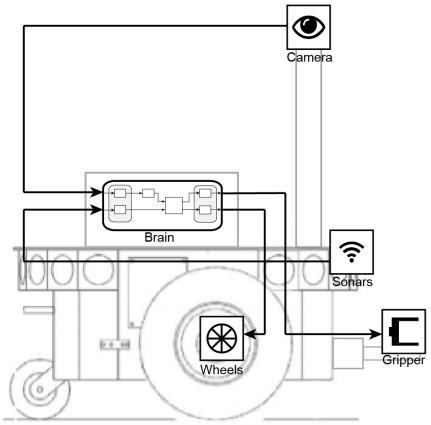

Figure 4.7: Diagram of the conceptual modules in a real life application. The virtual camera and sonars are substituted by the real senses and analogously with the virtual gripper and wheels and the real actuators. The brain is unchanged and executed inside the PC on the robot.

4.5 Real experiment

Finally, we performed a proof of concept of the applicability of the algorithm implemented in the simulation to a small experiment with the real robot. We used the algorithm for oriented navigation since it is the most simple task of the two and does not require the installation of the gripper.

We tested whether, leaving unchanged the algorithm, we could use it to instruct the robot what to do when encountered with a bifurcation: turn right (number), stop (shape) or turn left (letter).

In Fig 4.7 we see how the modules are conceptually set in a real life experiment. The brain is run in the PC on board and receives input from the camera and sonars and outputs commands to the wheels and gripper.

On the other hand, in Fig. 4.8 we see the Simulink model for the real experiment. If you reference back to Fig. 4.3, we have changed the virtual camera and sonar by modules that connect to the real camera and sonars of the robot (via ARIA) and the output of the brain connected to a module that sends commands to the robot's wheels (via ARIA as well) instead of to the simulated environment. The brain module of the robot is completely unchanged. We use the 'RealTimer' module to make the execution work in real time. The camera zoom and the sonar sensibilities of the robot had to be tuned to ensure the robot stopped at a good distance of the wall such that the symbol was not too small.

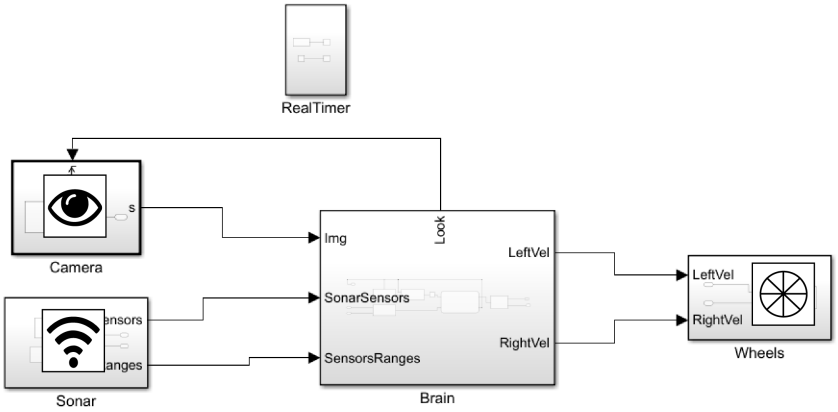

Figure 4.8: Simulink model for the real experiment. We change the virtual sensors for the real ones and introduce a module for the wheels. The brain module is left unchanged.

4.6 Results of the real experiment

As shown in Fig. 4.9 the robot approaches the wall and correctly executes the action demanded in each case by the symbol shown on the wall. These stills have been taken from a video available in the repository.

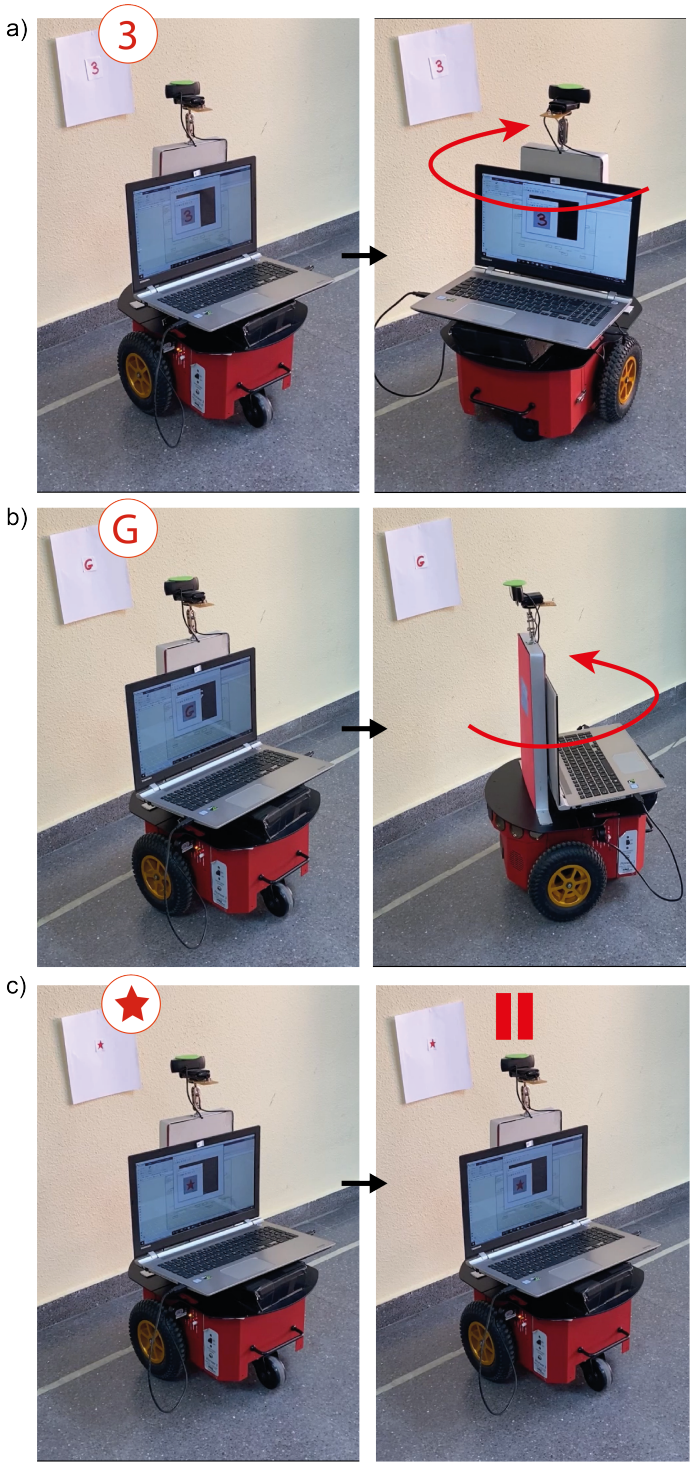

Figure 4.9: Experiment with the real robot. The robot approaches the wall and stops. With a number (3), the robot turns right; with a letter (G) it turns left and with a shape (Star) it stays still.

Chapter 5

Conclusions and future work

Is the model able to associate natural stimuli? What are the main constraints to take into account or to solve?

We have seen the model correctly associating complex stimuli. However, it would not be correct to say this association is direct since it depends on a big prior preprocessing that transforms these natural stimuli in some representation that the network works with. This seems common in biological systems that process the raw information before associating the stimuli (e.g. in the visual cortex).

We have studied in depth the importance of the orthogonality of the stimuli to avoid *the problem of difference*. In particular, using image moments as preprocessing, due to the fact that they are not invariant to perspective, made the system have some level of error. Assigning this error to the hippocampus model is wrong as in general it is expected to work optimally learning and retrieving, and the error rate should be assigned to the preprocessing module for not being able to properly generate the same representation for two different images of the same symbol. The fact that this could be solved by accounting perspective has been well justified. Perspective is a tricky transformation since in general you need metainformation of the environment to detect that you are taking an image with perspective. This means that if you see the segmented symbol there is no way to know if it has been perceived with perspective or not. As an example, if after segmentation the figure is an ellipse, in general there is no way to figure out if you have captured an ellipse with low perspective or the image of a circle with high perspective. This is why there are no invariant moments to perspective. However, we have solved this problem using a CNN for feature extraction and use the activation of one of the layers as representation. Indeed, we saw this representation was invariant to perspective (without the need of metainformation) and therefore solved very satisfactorily the problem.

In regards to the theoretical aspect of the model we see two main flaws. The first one is the necessity of perceiving the stimuli so many times, in a cyclic manner, for the selective layer to become selective to them. In particular we show the 9 images cyclically 33 times each. In real situations, biological systems are able to form episodic memories from stimuli that are presented once rather than being repeated over and over again. The second one is the big separation we do between learning and retrieval. Introducing inhibition in the

form of a supportive layer of inhibitory neurons helped greatly to solve the problem of lost stimuli. However, during retrieval, we tried to compute the activation to see if the system converged to some fixed point and, while it did, it was one with high selectivity but no concept cells. Therefore, we had to make a split between the two processes. This is an issue because biological systems do not separate from both tasks. Also it arises the question of how well the model would work if, after learning a number of associations, we tried to make it learn new ones. Further look into this is needed.

Can a mobile robot use the model for a significant application?

We have seen the model once it has learned works perfectly in retrieving when presented with stimuli similar to the one learned in the learning phase. We have seen as well the robot solve a number of tasks using this ability.

The biggest issues with the experiments where that they were toy experiments (more complex tasks with more complex associations and more symbols need to be explored) and that they where simulated. In regards of the latter, we have made the robot's algorithm modular and used the specifications of the real robot so the testing in a real life setting could be done as directly as possible. Thanks to this, the adaptation to the real robot was direct and only configuration and tuning of the sensors, camera and wheel velocities were needed. We saw a proof of concept that the system would work in a real life experiment.

Another aspect of our present project is that it has studied the ability of the model to learn offline and to retrieve in a real life situation when, in general, learning should also be tested online. We see here that many considerations arise and is a problem that should be tackled in the future on its own. To state a few, we show all the images of the training set cyclically 33 times. Having the robot move or a human agent intervene in order to be changing the images the robot it is capturing would be highly time consuming and unfeasible. A possible solution would be for the robot to take all the pictures and then go through the learning process in an offline manner (going cyclically through the pictures taken internally). Another issue is that the robot should communicate when it is trying to be taught a new stimuli that the preprocessing module considers too similar to a previous one, and therefore, would lead to *the problem of similarity*.

Finally, we have hardcoded what the robot should do when detecting different stimuli (e.g. a number means turn right) but it would be interesting to see whether the robot could assign the stimuli to the actions via some system of reward-punishment through reinforcement learning. This would make the association stimulus-action mimic more closely biological systems.

As a final conclusion we think the model despite of (or thanks to) its simplicity, works very well in performing the task it is intended to: associate stimuli. It needs, as it seems natural, the aid of a good preprocessing module to work properly in practical situations for artificial cognition. The study is in its early start but looks promising and more research is needed.

Bibliography

[1] Bellman, R. (1966). Dynamic programming. *Science 153*(3731), 34–37.

[2] Wang, Y., H. Yao, and S. Zhao (2016). Auto-encoder based dimensionality reduction. *Neurocomputing 184*, 232 – 242.

[3] Kainen, P. (1997). Utilizing geometric anomalies of high dimension: When complexity makes computation easier. In M. Kárný and K. Warwick (Eds.), *Computer Intensive Methods in Control and Signal Processing*. Birkhäuser, Boston, MA.

[4] Donoho, D. (2000, 01). High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, 1–32.

[5] Chen, D., X. Cao, F. Wen, and J. Sun (2013). Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 3025–3032.

[6] Murtagh, F. (2009, 12). The remarkable simplicity of very high dimensional data: Application of model-based clustering. *Journal of Classification 26*(3), 249–277.

[7] Gorban, A. N., I. Y. Tyukin, and I. Romanenko (2016). The blessing of dimensionality: Separation theorems in the thermodynamic limit. *IFAC-PapersOnLine 49*(24), 64 – 69.

[8] Gorban, A. N. and I. Y. Tyukin (2018). Blessing of dimensionality: mathematical foundations of the statistical physics of data. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 376*.

[9] Tozzi, A. (2019). The multidimensional brain. *Physics of Life Reviews 31*, 86 – 103.

[10] Megías, M., Z. Emri, T. Freund, and A. Gulyás (2001). Total number and distribution of inhibitory and excitatory synapses on hippocampal ca1 pyramidal cells. *Neuroscience 102*(3), 527 – 540.

[11] Goriounova, N. et al. (2018, 12). Large and fast human pyramidal neurons associate with intelligence. *eLife 7*, 1–21.

[12] Tyukin, I. Y., A. N. Gorban, C. Calvo Tapia, J. Makarova, and V. A. Makarov (2019). High-dimensional brain: A tool for encoding and rapid learning of memories by single neurons. *Bull Math Biol. 81*(11), 4856–4888.

[13] Calvo Tapia, C., I. Y. Tyukin, and V. A. Makarov (2020). Universal principles justify the existence of concept cells. *Sci Rep. 10*(1), 78–89.

[14] Barlow, H. (1972). Single units and sensation: A neuron doctrine for perceptual psychology? *Perception 1*(4), 371–394.

[15] Quian, R., L. Reddy, G. Kreiman, C. Koch, and I. Fried (2005, 07). Invariant visual representation by single neurons in the human brain. *Nature 435*, 1102–7.

[16] Quiroga, R. Q. (2012). Concept cells: the building blocks of declarative memory functions. *Nature Reviews Neuroscience 13*, 587–597.

[17] *Types of memory.* Queensland Brain Institute. https://qbi.uq.edu.au/brain-basics/memory/types-memory.

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

[19] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks 61*, 85 – 117.

[20] Sinz, F. H., X. Pitkow, J. Reimer, M. Bethge, and A. S. Tolias (2019). Engineering a less artificial intelligence. *Neuron 103*(6), 967 – 979.

[21] Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks 10*(9), 1659 – 1671.

[22] Hodgkin, A. L. and A. F. Huxley (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology 117*(4), 500–544.

[23] Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science 11*(1), 23–63.

[24] *Mobile Robot Simulation for Collision Avoidance with Simulink.* MathWorks. https://es.mathworks.com/matlabcentral/fileexchange/47208-mobile-robot-simulation-for-collision-avoidance-with-simulink.

[25] Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology 15*(3), 267—273.

[26] LeCun, Y., C. Cortes, and C. J. Burges. *MNIST database.* http://yann.lecun.com/exdb/mnist/.

[27] Martinsanz, G. P., J. M. de la Cruz García, and A. G. Tome (2001). *Visión por Computador: Imágenes digitales y aplicaciones.* RA-MA S.A. Editorial y Publicaciones.

[28] He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

[29] *ResNet-50*. MathWorks. https://es.mathworks.com/help/deeplearning/ref/resnet50.html#:~:text=ResNet%2D50%20is%20a%20convolutional,%2C%20pencil%2C%20and%20many%20animals.

[30] *ImageNet*. http://www.image-net.org.

[31] *Pioneer 3TM Operations Manual*. ActivMedia Robotics. http://alvarestech.com/temp/RoboAseaIRB6S2-Fiat/www.fit.hcmup.edu.vn/~hungnv/teaching/Robotics/Robo_HW_manual.pdf.

[32] *Pioneer 2 Gripper Manual*. ActivMedia Robotics. https://www.macalester.edu/research/fox/pioneer/gripmanP2.pdf.

Appendix A

Stateflows

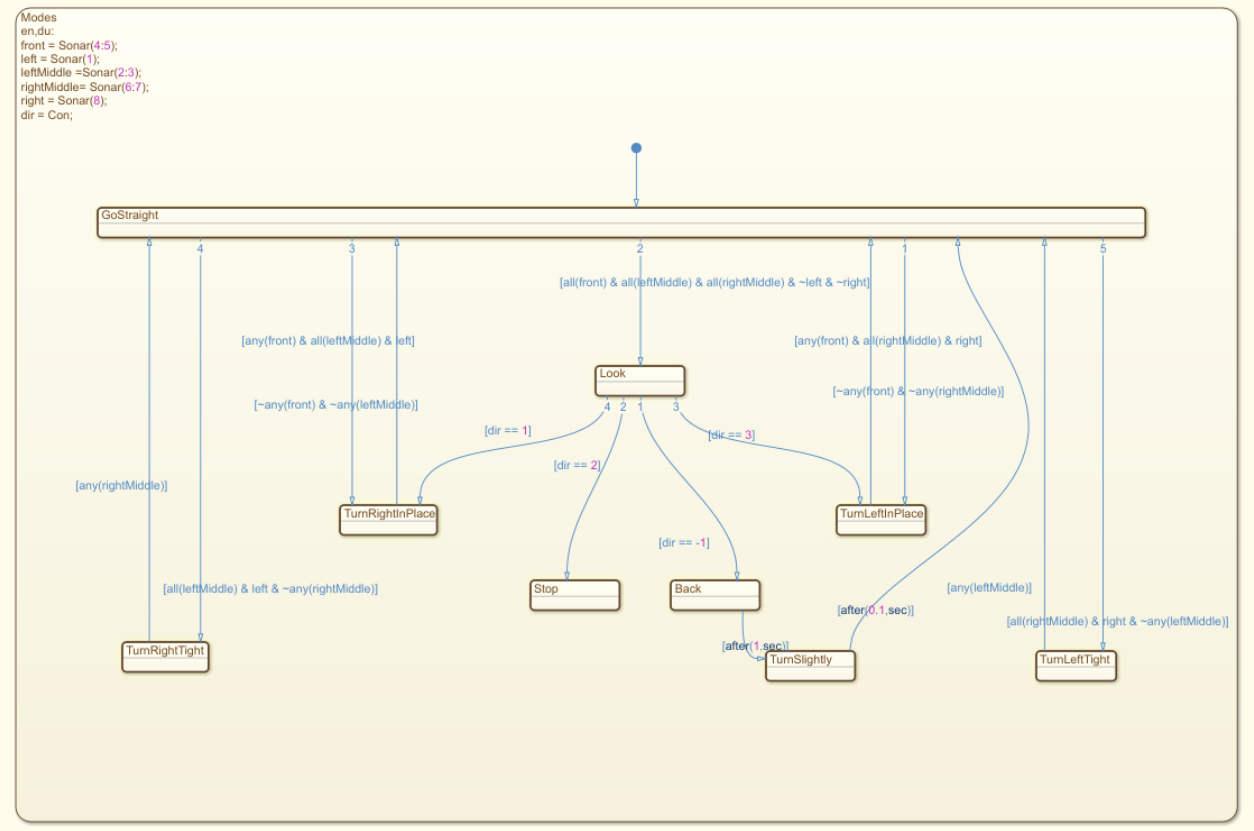

Figure A.1: Stateflow diagram used in the algorithm for the oriented navigation task.

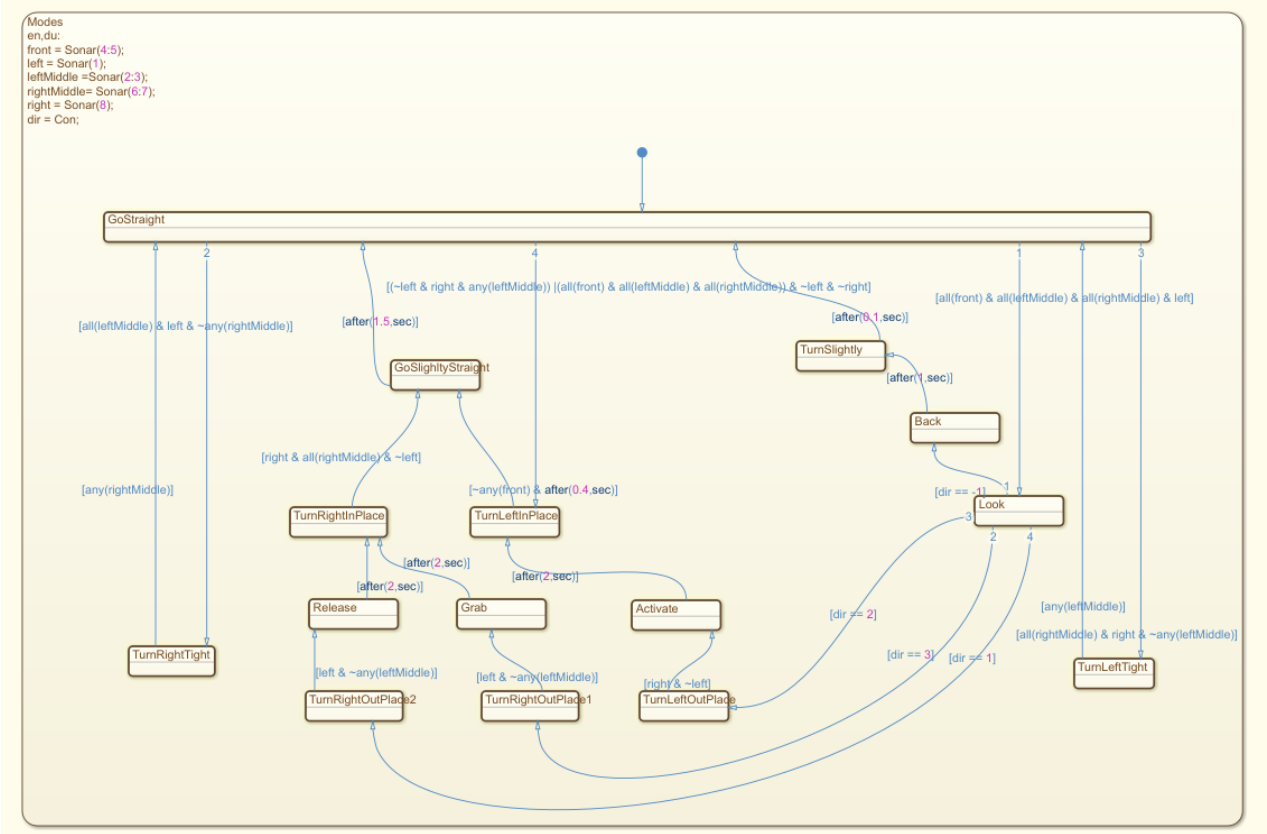

Figure A.2: Stateflow diagram used in the algorithm for the order execution task.