Dottorato di Ricerca in

Scienze Computationali e Informatiche

Ciclo XXII

Consorzio tra Università di Catania, Università di Napoli Federico II,

Seconda Università di Napoli, Università di Palermo, Università di Salerno

Sede Amministrativa:

Università degli Studi di Napoli Federico II

Francesco Donnarumma

A Model for Programmability and Virtuality in Dynamical Neural Networks

---

Tesi di Dottorato di Ricerca

Il COORDINATORE

Prof. Luigi M. Ricciardi

This work was partially written while the author was visiting Ecole Normale Supérieure De Cachan, supported by ESF GAMES project, short visit grant n. 3014

Contents

List of Figures

1 Introduction

1.1 Background and motivation

This thesis work proposes a *fixed-weight* neural network architecture, *Dynamical Multiplication Architecure* (DMA), which is built on a biologically plausible recurrent Artificial Neural Networks (ANNs) model - *Continuous Time Recurrent Neural Networks* (CTRNNs), which, possessing the *virtuality* capability, we demonstrate to be a suitable architecture in order to capture neuronal phenomena involving programmability. Note that:

> by 'programming' we mean the fact that a fixed structure – functionally identifiable with an interpreter – can be conditioned (programmed) by an auxiliary (programming) input so as to exhibit a repertoire of qualitatively different behaviors.

Natural, i.e. biological, phenomena, controlled by neuronal activity, are often modelled by Artificial Neural Network (ANN) architectures, both in lower level animals as well as in mammals and humans as attested by a wealth of successful accounts of

mental performances of a perceptive, motor, cognitive and, recently, also emotional nature (see, for instance Lambrinos et al., 2000; Riesenhuber and Poggio, 2002; E. and Oztop, 2002; Tani et al., 2004; Schindler et al., 2008; Bailu and Sil, 2009; Friston and Kiebel, 2009; Huo and Murray, 2009).

In most cases these ANN architectures implement "special purpose systems". In other words, the ANN is developed in such a way as to exhibit a *unique/special behaviour* in response to the input signals. In general, the behaviour of the ANN is obtained through the use of an appropriate learning algorithm, quite independent of the activity of the network that is being developed, which sets the network structure (e.g., the connection weights). In this way during the learning phase numerous different ANN structures are tested resulting in multiple different ANN behaviours. However, when the desired structure is reached, the ANN is able to perform a unique dynamical behaviour only, at the end realizing a specialized special purpose system.

However it cannot be denied that many, or at least some, crucial biological phenomena exhibit a substantial/qualitative change of behaviour in "short time" presumably without involving a neuronal connectivity changing. So these biological phenomena seem to be clamouring to be interpreted as *genuine computational* tasks, and such as to need *programming*. As is well known, functional modelling of biological phenomena by computational, algorithmic means has been the mainstay of Artificial Intelligence. However the word 'computation' is still being used in a variety of often misleading or poorly understood ways such as occurs within the "computational neuroscience" where 'computational' usually refers to some kind of

ANN used to model some part of the biological nervous system as mentioned before. Most of these usages are of a metaphorical nature. On the other hand we based the dissertation on a clearer definition of programmability, summarized above, concerning the possibility of building fixed interpretive structures exhibiting *different shapes of behaviour*[1] at varying an auxiliary input.

In the next Section a number of biological phenomena are sketched, which are chosen as witnesses to clues of the programmability phenomenon in neuronal structures.

1.2 Programmability in biological systems

Several biological and behavioural findings suggest that some kind of programming capability is needed. In order to clarify the kind of phenomena referred to in this thesis, in this section some cases where such capability seems to be needed are described.

Grid and Place cells in rats. Neuronal circuits constituting a distributed spatial map of the environment have been recently found in rats. These circuits are composed of the so-called 'Grid-cells' in the dorsocaudal medial entorhinal cortex and the 'place cells' in the dentate gyrus and in the hippocampal area CA3 of the rat. These cells exhibit the tendency to extensive "remapping" in response to changes in the sensory or motivational inputs. Remapping is expressed under some conditions as a change of firing rates, while under other condi-

[1] Here by 'shape of behaviour' it is denoted the sum total of the responses or behaviours of the ANN to external stimuli (data), be they instantaneous or prolonged. Thus, shape of behaviour can be taken as the overall mapping performed by the network.

tions as a complete reorganization of the hippocampal activity. In (Fyhn et al., 2007) the authors underline the capability of these circuits to switch from one map to another:

> "...to evoke instantaneous global remapping, the room lights were turned on after 11 min of running in the dark condition on the test day. In one animal this caused a sudden reversion to the original hippocampal map associated with the light condition..."

Natural Language Natural language performance and mental arithmetic involve the production of an a priori unlimited series of sentences and symbolic processes. The existence of specific, fixed structure modules responsible for each sentence or group of sentences as well as for each specific symbolic mental process seems to be out of the question given the increasing number of such units formed according to need, and, even if by retrofitting existing units through learning this number could be limited, still the time delays implied by learning would make the retrofitting hypothesis implausible. Recently Dehaene (Dehaene, 2005), has argued that the vast symbol processing capabilities of the human brain, including recursion involved in doing mental arithmetic, are explained neither in terms of evolutionary adaptation, nor of learning but in terms of some "recycling" of neurons. Moreover the notion that well defined topographical areas of the cortex are capable of switching "on the spot", as it were, between different behaviours – no learning phase – is marginally expressed in the neurobiological literature.

Theory of mind. Mind reading, or Theory of Mind, denotes the capability of predicting and recognizing the intentions of conspecifics and the consequent determination of their behaviour. Models for Theory of Mind in cognitive science suppose that prediction of behaviour in conspecifics is performed either by inferring from a description of their present behaviour a description of the "next state" of it (Theory theory) or by internally enacting a simulation (Simulation) of how their behaviour evolves. In both cases, some sort of forward or inverse model of the "other" is apparently needed, except in the most stereotyped and instinctual situations. Unless there exist specific modules dedicated to each behaviour of each individual, it appears that some definite neuronal areas are capable of interpreting sensorial data and, especially, of planning the appropriate ensuing behaviour by altering their functional responses - without learning - in consequence of the appraisal of the behaviour of the conspecific (Gallese and Goldman, 1998).

Area F5 in the macaque motor cortex. In the phenomenology of the neuronal activity in the F5 area of the macaque monkey's motor cortex, neuronal cells of this area selectively respond to the type or the modality of an object-directed action (e.g. "grasping an object" or "grasping an object by a precision grip"). Moreover some of these neurons show a clear selectivity for a specific phase of the action. In particular it has been proposed (Fadiga et al., 2000; Rizzolatti and Gentilucci, 1988) that a sort of "vocabulary" of movements is stored in the F5 area. The functional difference between the activity of F5 neurons and that of strictly involved neurons of the precentral motor cortex (F1) is that the first one is

a coding for object-directed actions (or fragments of object-directed actions), while the activity of F1 neurons is a coding for movements regardless of the action context in which they are performed. Then, in a manner equivalent to that stated for F5, neurons of area F1 could be defined as a "vocabulary of elementary movements". Also in this example, unless there exist specific F5 neuronal modules dedicated to each specific instance of an action category (hold, grasp, tear, manipulate), F5 area seems to be capable of changing its shape of behaviour "on the fly" and without changing its structure, i.e. without learning.

It thus appears that these neuronal phenomena do display a behaviour typical of a *multi-*, if not altogether *general purpose*, system and, therefore, should be interpreted as genuine computational tasks, involving some kind of *programming*.

1.3 A model for programmability in neural networks

So these among other phenomena controlled by neuronal activity, seem to suggest the implication of some form of programmability.

Then, the main original contribution of the thesis work is to face the following research question: *if biological phenomena need programming, how can they be modelled by biological plausible ANN architectures?*

This work reflects the assumption that fixed-structured neural network models can be used to give an account for these phenomena; so in order to give an answer to this question

this work proposes an architecture for *fixed-weight* Continuous Time Recurrent Neural Networks (CTRNNs) which has the following properties:

a. the ANN variables (neuron output, neuron input, activation function, etc.) have a direct biological interpretation;

b. the qualitative changes of behaviour are *controllable* in a *computational* sense. By this we mean not only that the auxiliary input signals causing a change of the shape of behaviour are in a well-defined and causal relationship with the change itself, but also that they are physically homogeneous with the I/O neural activity (see Fig. (1.1));

c. the fixed-weight ANN has the capability to exhibit a wide repertoire of qualitatively different shapes of behaviours depending on the auxiliary (programming) inputs.

In standard computational systems (e.g. Turing Machine, Von Neumann architecture) points b. and c. above are obviously achieved by the concept of *programmability.* There the input can be interpreted as *data* or *programs.* And the programs define the shapes of behaviour of the system. In (Garzillo and Trautteur, 2009) the authors propose that the concept of programmability can be expressed, even beyond the context of algorithmic computability, by the concept of *virtuality.* Then the ANN model proposed in this paper possesses the properties described in points b. and c. because it possesses virtuality. This is achieved by "pulling out" from the CTRNN the multiplication operation, usually used to model the input of biological neurons, by using subnetworks providing the outcome of the multiplication operation between the output coming from presynaptic connected neurons and the weights associated with

the connections. As a consequence the weights can be given as an auxiliary input to the original network augmented with the multiplication subnetworks, thus creating a neural architecture with *two kinds of input lines*: auxiliary (or programming) input lines in addition to standard data input lines. Therefore, the auxiliary input lines can be fed with a code describing the original network in a way resembling the code of a virtual machine in a standard computational architecture or the Gödel numbers given to a Universal Turing machine. In other words the code fed to a DMA network is in direct connection with the structure it *simulates* and so it can be viewed as an *interpreter* for a class of CTRNNs.

1.4 Related approaches

The *fixed-weight neural networks* rubric is a recent emerging area of research in the neural network field. The idea of fixed-weight neural networks seems to originate in (Cotter and Conwell, 1990). In this work, Cotter and Conwell separate the concept of *learning* (on a longer time scale) which implies the change of the weights, the structure of the network, from that of *adaptation*, which is the change of behaviour that the fixed structure can produce in response to different, or varying, types of inputs, with no persisting effect after a suitable rest period. In all the thesis work "fixed-weight networks" denote artificial neural networks for which the weights - the structure - are never varied in use. A number of approaches have been proposed which satisfy these definitional requirements (Younger et al., 1999; Hochreiter et al., 2001; Izquierdo-Torres et al., 2008; Zegers and Sundareshan, 2003). In some works about

fixed-weight neural networks (Hochreiter et al., 2001; Zegers and Sundareshan, 2003; Ito and Tani, 2004) the authors also consider two kinds of input lines, one that carries data, which the network is demanded to respond to, the other which carries information identified as an auxiliary *context* input, on the basis of which, the network can correctly respond to the data presented.

In other works (Maass et al., 2002; Steil, 2004), this clear separation between two lines is not present, though great emphasis is laid on how the use of a fixed structure, a universal kernel or reservoir, can capture different dynamics and behaviours, in a general and robust way. In a similar context another interesting work (Izquierdo-Torres and Harvey, 2007; Izquierdo-Torres et al., 2008) tries to teach a fixed structure to emulate a sort of Hebbian learning rule only by changing the input current signals, thus intimating questions on whether this phenomenon should be treated as learning, adaptation or other, and where the stored information vanishes when non-input signal is given.

As it will appear clearer in Chapter 3, even though this work is framed in the fixed-weight rubric, it substantially differs from these approaches both in its goal and in its implementation.

Moreover in the field of unconventional or alternative computing, a number of systems not immediately computational in the algorithmic sense, such as neural networks, are shown to be capable of (at least) *simulating*, in some well-defined sense, a generic Turing machine, thus including a Universal Turing machine which is the theoretical underpinning of virtuality. The universality property and therefore virtuality is correctly assigned to those systems. In particular such is the case for a

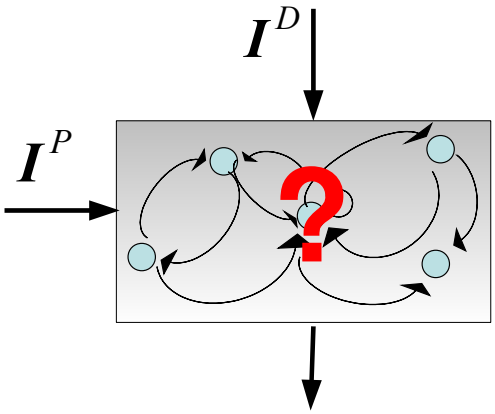

Figure 1.1: Searching for a neural network structure possessing the programmability/virtuality capability. In this thesis an original architecture is proposed, DMA, capturing virtuality and in this sense resulting in a good model for biological phenomena related to programmability. Note, as the figure suggests, that this architecture should be able to process on the same level data I^D and programs I^P resembling properties of computational devices.

dynamical system simulation of algorithmic devices (Graça et al., 2005; Branicky, 1995) and for a universal ANN implementing a Universal Turing machine (Siegelmann and Sontag, 1995; Siegelmann, 1999).

However, in this thesis, in the spirit of biologically plausible modelling, the aim is not to simulate other computational systems (Turing machines), but rather to search for virtuality within the model itself, which might be called *CTRNN* or *material virtuality*, in the hope that this capability might be transferred to, or rather discovered in, the biological reality.

1.5 Plan of the dissertation

In Chapter 2 a step into the CTRNN model chosen for the substrate of DMA is taken, motivating its choice. Moreover the possibility of studying its properties in the frame of the dynamical system theory will pave the way to the analysis of networks behaviour developed in the rest of the thesis work.

In Chapter 3 the virtuality capability is presented as the proper strategy to provide a dynamical system with programmability. Inspiring literature is examined and a robotic case study is driven motivating the recourse to virtuality in our approach.

Chapter 4 is the core chapter in which the original DMA of fixed-weight neural networks is developed. We explain how this architecture is endowed with virtuality / programmability, thus capturing programmability capability in a broader sense.

In Chapter 5 the methods used for comparing DMA networks are shown, developing the theoretical background that will be widely deployed in the realization of the tests. Notice that some of these theoretical results, necessary for the analysis of DMA, are for the first time in literature introduced for neural networks studied as dynamical systems, bringing to this dissertaion a side effect original contribution.

In Chapter 6 a number of experiments is performed, showing how inside the proposed architecture interpreters can be programmed with auxiliary inputs in order to reproduce the dynamical behaviours of networks coded by the auxiliary input. On the other hand a study of the robustness of the proposed architecture with respect to variations of the I/O time scales is provided.

In Chapter 7 the conclusions on the presented approach are drawn, together with underlying results, envisaging possible

future work on this line of research.

2

CTRNNs as models of neuronal networks

In this Chapter a particular model of Artificial Neural Networks will be analyzed in order to develop in Chapter 4 the DMA which this thesis proposes.

Artificial neural networks (ANNs) are computational models implemented in software or specialized hardware devices that attempt to capture the behavioural and adaptive features of biological nervous systems. They are typically composed of several interconnected processing units, nodes or 'neurons' (see Fig. 2.2) which can have a number of inputs and outputs.

This architecture tries to mimic systems of biological neurons, the basic information processing elements in the Central Nervous System (CNS). These elements are able to perform at a high degree of parallelism complex processes that Artificial Intelligence has been trying to emulate since its birth.

In mathematical terms, an ANN can be seen as a directed graph where each node implements a neuron model. Several models have been proposed since the first one of McCulloch and Pitts (McCulloch and Pitts, 1943). In the simplest case, the neuron model computation is just a weighted sum of the incoming signals transformed by a (typically nonlinear) static trans-

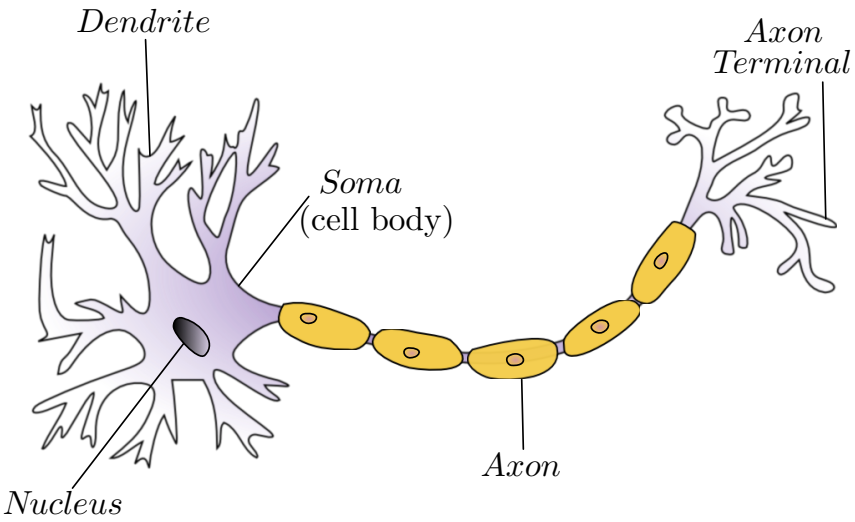

Figure 2.1: A schematization of a biological neuron

fer function; however more sophisticated neuron models involve discrete-time or continuous-time dynamics (like the model chosen in this work). The connection strengths associated with the edges of the graph connecting two neurons are referred to as synaptic weights, and the neurons with connections to or from the external environment are often called output or input neurons, respectively. The number and type of neurons and the set of possible interconnections between them define the architecture or topology of the neural network.

In this work the underlying model of the DMA consists of Continuous Time Recurrent Neural Networks (Hopfield and Tank, 1986).

The choice of *continuous time* fits the desire of modelling natural systems, i.e. physical or biological systems, which involve inherently continuous time.

The word recurrent stands for the possibility of the presence of loops inside the direct graph topology. Unlike static feed

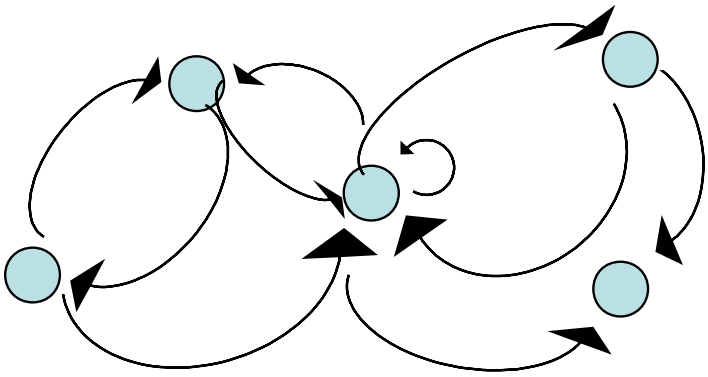

Figure 2.2: A direct graph can in general capture a general model for an artificial neural network. The main difference between models is given by the implementation of each node of the graph into a different neuron model. The more biologically accurate this representation is, the more complex these models become.

forward networks, recurrent neural networks (RNNs) are said to be dynamic because the presence of feedbacks allows time-dependent behaviours.

Though this artificial neuron model is more elementary than other more accurate models (see Abbott and Kepler, 1990; Izhikevich, 2004), a CTRNN model is a very attractive abstraction of the biological network because (Kier et al., 2006):

- the CTRNN neuron has a plausible biological interpretation;
- it is computationally inexpensive to implement;
- CTRNNs are universal dynamics approximators: any trajectory of a smooth dynamical system can be approximated to any desired degree of accuracy by these systems with a sufficient number of nodes(Funahashi and Nakamura, 1993).
- the model is a mathematically tractable system: some works studying their dynamics exist (e.g. Beer, 1995b, 2006);

In the following sections these statements will appear clearly motivated.

2.1 CTRNN model

Continuous Time Recurrent Neural Networks (CTRNNs) are networks of biologically inspired neurons described by the following general equations (Hopfield and Tank, 1986; Beer, 1995b):

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{N} w_{ij}\sigma(y_i - \theta_i) + I_i^e \qquad i \in \{1, \ldots, N\} \quad (2.1)$$

where N is the number of neurons in the network and for each neuron i:

- τ_i is the *membrane time constant*,
- y_i is the *membrane potential* after the deletion of the *action potential*,
- θ_i is the *threshold*,
- $x_i = \sigma(y_i - \theta_i)$ is the *mean firing rate*, with $\sigma(x)$ the *activation function*,
- $I_i^e = \sum_{j=N+1}^{N+L} w_{ij}x_j$ is an *external input current* coming from L external sources x_j
- w_{ij} is the *synaptic efficacy* (*weight*) of the connection coming from the neuron j

In general the function $\sigma(x)$ can be any *smooth*, *monotonic* and *bounded* activation function. For example we could use the parametric form (Tino et al., 2001)

$$\sigma_{a,b,c}(x) = \frac{a}{1 + e^{-c \cdot x}} + b$$

which has the advantage of reducing to the hyperbolic tangent function with $a = 2$,$b = -1$, $c = 2$

$$\sigma_{2,-1,2}(x) = \frac{2}{1 + e^{-2 \cdot x}} - 1 = \frac{e^x}{e^x} \cdot \frac{1 - e^{-2 \cdot x}}{1 + e^{-2 \cdot x}} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x)$$

or the standard sigmoid with $a = 1$, $b = 0$, $c = 1$

$$\sigma_{1,0,1}(x) = \frac{1}{1 + e^{-x}}$$

This choice of sigmoid activation function has been made in all the applications of Chapter 6. However, the approach is quite general so that different activation functions respecting the smoothness, monotonic and boundedness properties could allow similar results.

2.2 CTRNN biological interpretation

Compared with more biologically-realistic neural models, the dynamics of an individual CTRNN neuron is quite trivial. However, small networks of CTRNNs can qualitatively reproduce the full range of nerve cell phenomenology, including spiking, plateau potentials, bursting, etc. (Beer, 2006).

Continuous Time Networks were proposed in (Hopfield and Tank, 1986) deriving their inspiration from considering the neuronal membrana as modelled by an RC circuit (see Fig. 2.3), and thus obtaining for the membrane potential V_M the equation

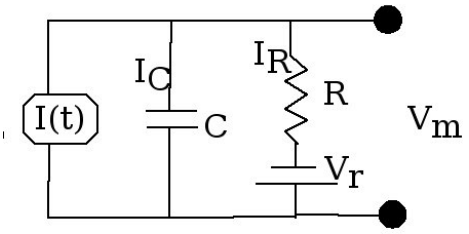

Figure 2.3: The RC circuit schematization of the neuronal membrane.

$$C_M \frac{dV_M(t)}{dt} = I_R(t) - \frac{V_M(t)}{R_M} + I(t) \tag{2.2}$$

These equation describes the charging of the membrane capacity C_M by the sum of three sources: postsynaptic currents I_R induced by presynaptic activity directed to the neuron, leakage current due to the finite input membrane resistance R_M and input currents $I(t)$ from other neurons external to the circuits. The analogy with Equation (6.2) is clear, setting the time constant of the membrane $\tau_M = C_M \cdot R_M$

So the model having a clear direct counterpart in the variable of the biological neurons is still an intriguing one when creating models of biological neuronal networks, and in various works the model is considered so accurate as to make predictions on variables of a biologically modelled network, as for example in (Dunn et al., 2004) where CTRNN connectivity is used to suggest new functionalities for previously identified connections in the *Caenorhabditis Elegans*.

2.3 CTRNN and DTRNN

The CTRNN equations form a complex system of differential equations. In general, a complete quantitative description of the behaviour of the system is not possible. There are few cases in which a complete qualitative description is possible, as it is shown in the Subsection (5.1.2) for a single CTRNN neuron. Even stability analysis of the trajectories defined by the underlying CTRNN systems provides in many cases accurate descriptions in Dynamical System Theory as it will be clear in this Chapter.

However, in order to provide an implementation of CTRNN

trajectories a numerical integration of CTRNN equations should be supplied. Even though there are various accurate numerical integration methods for first order differential equations (see e.g. Strogatz, 1994), in this work Forward Euler method is chosen.

For an equation $dx/dt = f(x)$ the Forward Euler method allows a dicrete approximation of its solution:

$$x_{n+1} = x_n + f(x_n)\,\Delta t$$

for sufficently *small* time steps Δt.

Applying this formula to Equation (6.2) we obtain the following discrete-time version of the neuron update equation

$$y_i^{n+1} = y_i^n + \frac{\Delta t}{\tau_i}\left(-y_i^n + \sum_{j=1}^{N} w_{ij}\sigma_j\left(y^n - \theta_j\right) + I_i^e\right)$$

which can be treated as an independent model of Discrete Time Recurrent Neural Networks (DTRNN).

On the other hand suitable choices of Δt (Hines and Carnevale, 1998) result in a suitable approximated version of CTRNN. In the experiments in Chapter 6 all the trajectories were realized using this discrete approximation.

2.4 CTRNNs as universal approximators

The fact that CTRNNs are universal dynamical approximators is often cited to evoke the power of the CTRNN model. In fact the following theorem asserts that any solution of a set of Ordinary Differential Equations can be approximated as accurately

as wanted, by a suitable CTRNN with N neurons. The proof of this theorem can be found in (Funahashi and Nakamura, 1993).

Theorem 2.4.1. *(Funahashi - Nakamura) Given a Continuous Autonomous dynamical system D defined by P differential equations $\dot{\mathbf{x}} = f(\mathbf{x})$ and the initial conditions $\mathbf{x}(0) = \mathbf{x}_0$ for which a solution flow $\gamma_D(t, \mathbf{x}_0)$ exists, then $\forall \epsilon > 0$ there exists a CTRNN of N equations with variables $\mathbf{y}$, a proper initial condition $\mathbf{y}_0$ and $P \leq N$ "observable" output neurons $\mathbf{y}^P = \{y_1, \ldots, y_P\}$, such that for the trajectory solution flow restricted to the output neurons $\gamma^P_{CTRNN}(t, \mathbf{y}_0)$, it stands that*

$$\max_{t \in T} \left\| \gamma_D(t, \mathbf{x}_0) - \gamma^P_{CTRNN}(t, \mathbf{y}_0) \right\| < \epsilon$$

This powerful approximation theorem (with its extensions, see e.g. for non-autonomous systems Kambhampati et al., 2000), has a very interesting machine learning effect: if we are going to approximate a trajectory of a dynamical system we can always find a suitable CTRNN structure approximating it.

Notice that as in this thesis we need methods which allow to compare fixed-weight CTRNN behaviours, this theorem has some limitations for our purpose: in fact it formulates the dynamic equivalence for one solution at a time, while we are interested in the dynamical system in its whole and the *family* of solutions which it expresses.

For this reason Bisimulation techniques will be treated in Chapter 5, where also a reformulation of this theorem will be given.

2.5 CTRNN as dynamical systems

This Section is not meant to be an exhaustive treatment of dynamical system theory, for which it is possible to consult a huge literature of books (see e.g. Guckenheimer and Holmes, 1986; Hale and Koçac, 1991; Strogatz, 1994). Some basilar definition can be found in Appendix A. However a collection of definitions and specialized results for CTRNN framework are provided in a self-contained manner in order to pave the way to the treatment in Chapter 5.

2.6 Background notions on Dynamical System Theory

This section starts from the formal notion of dynamical system which brings together continuous and discrete time dynamical systems. Moreover elements for stability analysis and language which will be used overall the thesis work are introduced.

Definition 2.6.1. An *Autonomous Dynamical System* D is a 3-ple (X, γ, T) where

- X is a topological space named *State Space*
- T is the *Time Set*
- the *Flow* γ is an evolution function

$$\gamma \, : \, (t, x) \in T \times X \longrightarrow X$$

(X, γ) satisfies the semigroup properties for X, in fact writing $\gamma^t(x_0) = \gamma(t, x_0)$

- $\gamma^0 = \mathbb{I}$ is the identity $(\gamma^t(x_0) \circ \gamma^0(x_0) = \gamma^t(\gamma^0(x_0)) = \gamma^t(x_0))$
- Associativeness: $(\gamma^t \circ \gamma^s) \circ \gamma^r = \gamma^t \circ (\gamma^s \circ \gamma^r)$ from the definition $\gamma^{t+s} = \gamma^t \circ \gamma^s$

Definition 2.6.2. A *Continuous Autonomous Dynamical System* D is a dynamical system (X, γ, T) where $\gamma : (t, \mathbf{x}) \in T \times X \longrightarrow Y$ is given by the solution of the set of first order Ordinary Differential Equations (ODE)

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}) \tag{2.3}$$

Theorem 2.6.3. *Globally existence and unicity of solutions (Cauchy - Lipschitz).*

The solution of a system of differential equations with given initial conditions

$$\begin{cases} \mathbf{x}(0) = \mathbf{x}_0 \\ \dfrac{d\mathbf{x}}{dt} = f(\mathbf{x}) \end{cases} \tag{2.4}$$

exists, is global and unique if f is uniformly Lipschitz continuous

Proposition 2.6.4. *Global Existence and Uniqueness of solutions for a* CTRNN.

Proof. A CTRNN system respects definition 2.3. From Equation (6.2) $f_i(y) = -y_i + \sum_{j=1}^{N} w_{ij}\sigma(y_i - \theta_i) + I_i^e$. As the function σ is smooth $\sigma \in C^\infty(\mathbb{R}, \mathbb{R})$. Moreover with $\sigma'(y) < \frac{1}{4}$ we calculate

$$f_i'(y) = -1 + \sum_{j=1}^{N} w_{ij}\sigma'(y_i - \theta_i) + I_i^e < -1 + \sum_{j=1}^{N} w_{ij}/4 + I_i^e = M_i$$

resulting in $\sup\{f_i'(y)\} < M_i < \infty$ and $\sup\{f'(y)\} < \max(M_i) < \infty$. So for Proposition (A.1.14) f is uniformly Lipschitz continuous and from Theorem (2.6.3) sigmoid CTRNN equations have a unique global solution. □

Definition 2.6.5. The *Phase Space* of an N dimensional system with variables $\{x_i\}_{i=1}^N$ is the space in $\mathbb{R}^N$ in which the i-th coordinate is the value of a variable x_i.

Definition 2.6.6. The *trajectory through* $\mathbf{x}_0$ of a solution $\gamma(t, \mathbf{x}_0)$ of a dynamical system $D = (X, \gamma, T)$ is the set

$$\{(t, \gamma(t, \mathbf{x}_0) \in T \times X \,:\, t \in T\}$$

Definition 2.6.7. The *orbit through* $\mathbf{x}_0$ of a solution $\gamma(t, \mathbf{x}_0)$ of a dynamical system $D = (X, \gamma, T)$ is the set

$$\{(\gamma(t, \mathbf{x}_0) \in X \,:\, t \in T\}$$

Definition 2.6.8. The *orbit structure* of a system is the number of the orbits and the direction of the flow on the orbits.

Definition 2.6.9. A *Fixed* (or *equilibrium*, *singular*, *stationary*) *Point* $\bar{x}$ of a dynamical system (X, γ, T) is a point such that

$$\gamma(t, \bar{\mathbf{x}}) = \bar{\mathbf{x}} \quad \forall t \in T$$

Definition 2.6.10. A *forward invariant* set S for a dynamical system $D = (X, \gamma, T)$ is a set for which $\forall \mathbf{x} \in S\ \gamma(t, \mathbf{x}) \in S \forall t \in T, t \geq 0$

Definition 2.6.11. An *invariant* set S for a dynamical system $D = (X, \gamma, T)$ is a both forward and backward invariant set, i.e. $\forall \mathbf{x} \in S\ \gamma(t, \mathbf{x}) \in S\ \forall t \in T$.

Definition 2.6.12. A set $S \subseteq X$ is *Attracting* for a dynamical system $D = (X, \gamma, T)$ if it is forward invariant and there exists a $\delta > 0$ such that $\forall \bar{\mathbf{x}} \in S, \mathbf{x}_0 \in X$ such that $\|\gamma(0, \mathbf{x}_0) - \bar{\mathbf{x}}\| < \delta$ then $\lim_{t \to +\infty} \gamma(t, \mathbf{x}_0) \in S$.

Note 2.6.13. In other words, any trajectory starting within a distance δ of a point of an attracting set is guaranteed to converge to the attracting set eventually (Notice that the trajectory can also escape from the neighbourhood of radius δ).

Definition 2.6.14. The *Basin of Attraction* B of an attracting set S for a dynamical system $D = (X, \gamma, T)$ is

$$B = \{\mathbf{x} \in X \ : \ \lim_{t \to +\infty} \gamma(t, \mathbf{x}) \in S\}$$

Definition 2.6.15. A set $S \subseteq X$ is *Globally Attracting* for a dynamical system $D = (X, \gamma, T)$ if the basin of attraction $B(S) = X$

Definition 2.6.16. A set $S \subseteq X$ is *Lyapunov Stable* for a dynamical system $D = (X, \gamma, T)$ if it is forward invariant and for each $\epsilon > 0$, there is a $\delta > 0$ such that $\forall \bar{\mathbf{x}}_1 \in S, \mathbf{x}_0 \in X$ such that $\|\gamma(0, \mathbf{x}_0) - \bar{\mathbf{x}}_1\| < \delta$ then $\forall t > 0 \exists \bar{\mathbf{x}}_2 \in S \ \|\gamma(t, \mathbf{x}_0) - \bar{\mathbf{x}}_2\| < \epsilon$.

Note 2.6.17. In other words, trajectories that start within a distance δ from an attracting set remain within a distance ϵ from the fixed point for all positive time (Notice that it is not guaranteed to converge to the attracting set).

Definition 2.6.18. A set $S \subseteq X$ is (*Globally*) *Asymptotically Stable* for a dynamical system $D = (X, \gamma, T)$ if it is both (globally) attracting and Lyapunov stable.

Definition 2.6.19. A set $S \subseteq X$ is *Neutrally Stable* for a dynamical system $D = (X, \gamma, T)$ if it is Lyapunov stable but not attracting.

Note 2.6.20. For example the equilibrium point of the simple harmonic oscillator is neutrally stable.

Definition 2.6.21. An *attractor* A for a dynamical system $D = (X, \gamma, T)$ is an invariant set for which $\forall \epsilon > 0$, $\forall \mathbf{x}_0 \in A$ $\forall \bar{\mathbf{x}} \neq \mathbf{x}_0, \bar{\mathbf{x}} \in A$ such that $\|\gamma(0, \mathbf{x}_0) - \bar{\mathbf{x}}\| < \epsilon$ then $\lim_{t \to +\infty} \gamma(t, \mathbf{x}_0) \in A$ (topological transitivity) - an attractor is attracting for a neighbourhood of itself.

Theorem 2.6.22. *For an Autonomous Dynamical system* (2.3) $D = (X, \gamma, T)$ *expressed by a System of ordinary differential equations the condition for fixed points* $\gamma(t, \bar{\mathbf{x}}) = \bar{\mathbf{x}}$ *becomes*

$$f(\bar{\mathbf{x}}) = \mathbf{0}$$

Theorem 2.6.23. *(Lyapunuv's indirect method) Stability classification for a dynamical system* (2.3)*. Given the Jacobian matrix* $J(\bar{\mathbf{x}})$ *in a fixed point* $\bar{\mathbf{x}}$ *and* λ_i *the corresponding eigenvalues then the fixed point results*

- *asymptotically stable if the real parts* $Re(\lambda_i)$ *of all the eigenvalues of* $J(\bar{\mathbf{x}})$ *are inferior to zero*
- *unstable if at least one real part* $Re(\lambda_i)$ *of an eigenvalue of* $J(\bar{\mathbf{x}})$ *is superior to zero.*

A *Hyperbolic Fixed point* $\bar{\mathbf{x}}$ is a fixed point for a Dynamical System $D = (X, \gamma, T)$ defined as in (2.3) if the eigenvalues of the Jacobian matrix computed in $\bar{\mathbf{x}}$, $J(\bar{\mathbf{x}})$, have non-zero real parts.

Theorem 2.6.24. *Maximum CTRNN fixed points number (Beer, 1995b) A CTRNN of N neurons can exhibit a maximum number of 3^N fixed points.*

The elements of stability theory introduced so far show how in some cases short and long term behaviour of CTRNNs can be studied starting from the weight values.

In the next section it will be exposed how CTRNN systems can be deployed in order to define mappings by means of the so-called *attractor computation*.

2.7 Attractor computation

In literature there are different ways of defining the input-output mapping of dynamical neural networks and in particular of CTRNNs (see for example Gupta et al., 2003).

The most common input choices are either the initial condition $\mathbf{y}(0) = \mathbf{y}^0$ or the external input currents $\mathbf{I}^e(t)$. If we consider the input for a group of neuronal cells corresponding to external signals coming from sensorial processes or from other neural groups, then, in CTRNNs, it is more biologically plausible to consider as input only the external input currents $\mathbf{I}^e(t)$. Otherwise the choice of considering as input the initial condition would imply that the external signals should be able to "overwrite" the membrane potentials of the neuronal cells.

The most common output choices are the trajectories, solutions of (6.2), or the steady state of the network − sometime referred to as *attractor computing* (Hopfield and Tank, 1985; Siegelmann et al., 2000). In the particular case in which 6.2 is globally stable, i.e. it has a unique stable fixed point independently of the initial condition $\mathbf{y}^0$, considering as output

the steady state of the network, the CTRNN implements the function:

$$\mathbf{f} \; : \; \mathbf{I}^e \in \mathbb{R}^N \longrightarrow \mathbf{f}\left(\mathbf{I}^e\right) \equiv \bar{\mathbf{s}}\left(\mathbf{I}^e\right) \in \mathbb{R}^N$$

where $\bar{\mathbf{s}}(\mathbf{I}^e)$ is the fixed point relative to the input $\mathbf{I}^e$**,** supposing that the time scale of the approach to the stable point is so fast as to make its computational delay negligible with respect to the time scale of $\mathbf{I}^e$. This approach was followed in building the CTRNN mul^* (See Sections 4.1 and 6) which *computes* a multiplication function.

3

Virtuality and Programmability

In this Chapter the programmability "strategy" which was chosen in order to realize the programmable DMA in Chapter 4 will be presented.

Features which capture the concept of programmability are needed in order to be "discovered" in continuous time dynamical systems like CTRNNs. So starting from approaches in literature, and driving a robotic experiment, it is shown how the properties a., b. and c. exposed in Chapter 1 are directly related to virtuality; thus virtuality, which is the hallmark of programmability, is motivated as the proper capability needed to implement programmable networks. Consequently the DMA will let us build programmable networks which are programmable because they possess the virtuality capability.

3.1 Learning and programmability in fixed neural structures

The construction of neural networks capable of shaping their behaviours according to input signals is quite challenging and

interesting in the arena of artificial neural networks with many potential applications: e.g. in autonomous robotics (Nishide et al., 2009; Salmen and Plöger, 2005), optical neural hardware (Younger and Redd, 2008), memorizing musical sequences (Eck, 2006), automatic speech recognition (Skowronski and Harris, 2007), natural language applications (Tong et al., 2007), machine learning (Jaeger et al., 2007) and computational models of biological systems (Yamazaki and Tanaka, 2007; Tani et al., 2004).

The usual way of deploying artificial neural networks is to find their structure by some kind of learning, designing what will be used as a special purpose system cabled for a specific task. Thus learning is usually associated with slow time scale synaptic plasticity modifications, but yet a line of research, which can be summarized as "learning with fixed weights", keeps pursuing some kind of non-synaptic plasticity since Yamauchi and Beer in 1994 challenged this view in (Yamauchi and Beer, 1994), where the authors described the abilities of fixed synapse continuous time recurrent neural networks (CTRNNs) to display reinforcement learning-like properties by exploiting internal network dynamics. Then more works try to develop fixed-weight networks showing clues to different shapes of behaviour: examples can be found in (Blynel and Floreano, 2003), where a dynamic network capable of "learning" without changing its weights in order to find food inside a T-maze is evolved with an evolutionary technique involving different epochs to perform incremental learning; in (Izquierdo-Torres et al., 2008), where a fixed-weight dynamic network simulating the adaptation capabilities of the *Caenorhabditis elegans* (temperature preference task) is presented; in (Izquierdo-Torres and Harvey,

2007), in which a fixed-weight dynamic network which is capable of performing Hebbian learning-like behaviour is shown. In this case learning is shown to arise from the interaction between *multiple timescale dynamics*:

> "Fast-time dynamics alter the slow-time dynamics, which in turn shapes the local behavior around the equilibrium points of the fast components by acting as a parameter to them".

Although there are differences between learning and programmability, as it will be pointed out in Chapter 7, it is worth driving a robotic experiment in order to point out the inherent problems of this approach and envisage a property solution: to endow a system with virtuality.

3.2 A preliminary study: Searching for virtuality in CTRNNs

First of all in order to explore the possibility of finding programmability in CTRNNs, we made a preliminary study in which we created a robotic controller with a system of CTRNNs resulting in a fixed structure trained in order different shapes of behaviours.

Analogously to (Younger et al., 1999; Hochreiter et al., 2001) we explicitly consider two kinds of input lines are considered in this architecture: one that carries *data*, which the network is demanded to respond to, the other which carries information the *context*, that we will identify as the *program.* This experiment was presented in (Donnarumma et al., 2007).

Building on (Paine and Tani, 2004) we set up a simula-

tion experiment concerning the switching between different response functions using the following scenario:

- a 2D mobile robot with 10 sonars as sensors and controlled by two parameters: the linear velocity v and the angular velocity ω around the vertical axis;
- a multiple $T-$maze as the robot environment;
- a twofold task of the robot: a) go forward along a corridor while avoiding possible obstacles, and b) turn left (right) at the next $T-$junction if a right (left) turn had been previously chosen. In other words we want the robot to proceed in the multiple $T-$maze through an alternation of turning choices.

Such task can be described by the following simple pseudocode:

```
BEGIN
leftTurn ← TRUE
WHILE (TRUE)
      IF (leftTurn = TRUE)
            leftTurn ← behaviour-
Right()
      ELSE
         leftTurn ← behaviourLeft()
      ENDIF
END WHILE
END
```

where behaviourRight() is a function (*program*) which controls the robot so as to make it a) go forward avoiding obstacles, b) turn right at a $T-$junction and return FALSE value; behaviourLeft() acts symmetrically.

Equivalently we give a CTRNN architecture, capable of running the two different response functions/programs - behaviour-

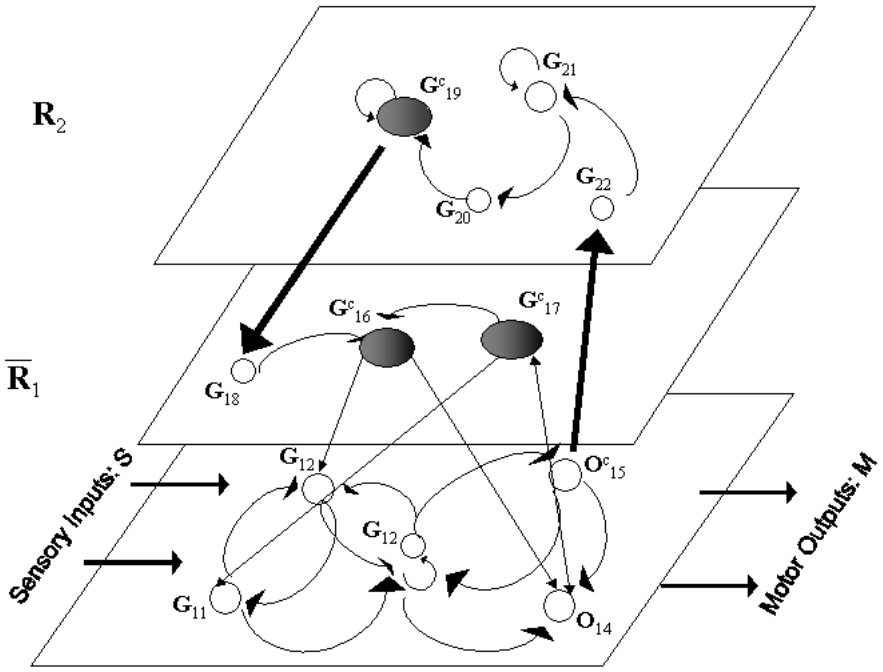

Figure 3.1: CTRNN architecture cabled for $T-$maze task.

Right() and behaviourLeft() - on a fixed subnetwork, which controls the robot in order to achieve the above task. Developing ideas by Tani (Paine and Tani, 2004), we *cabled* a two layered network (see Figure 3.1):

First layer. The first layer, L_1, is made up of seven neurons, that are initially fully inter-connected to each other. Five of these neurons directly receive input connections from the $K = 10$ sonars (the $\mathbf{I}^D$ data inputs). Two of these five neurons control the parameters v and ω, the linear and the angular velocity of the robot respectively. The remaining two receive the $\mathbf{I}^P$ program inputs. By setting the values on this last input, this sub-network is capable of controlling the robot in *two different*

Type	θ_i	w_{ij}	w_{ik}	τ_i	I_k^D	I_h^P	Total
Number	7	49	50	7	10	2	125

Table 3.1: Parameters for the first layer L_1 neural network equations

ways, selecting the *two different programs*: beviourRight() and behaviourLeft().

The equations of the layers are

$$\begin{cases} \frac{dy_1}{dt} = f(\{y_j\}_{j=1}^{N=7}, \tau_1, \{w_{11,j}\}_{j=1}^{N=17}, \theta_{11}, \{I_k^D\}_{k=1}^{K=10}) \\ \frac{dy_2}{dt} = f(\{y_j\}_{j=1}^{N=7}, \tau_2, \{w_{12,j}\}_{j=1}^{N=17}, \theta_{12}, \{I_k^D\}_{k=1}^{K=10}) \\ \frac{dy_3}{dt} = f(\{y_j\}_{j=1}^{N=7}, \tau_3, \{w_{13,j}\}_{j=1}^{N=17}, \theta_{13}, \{I_k^D\}_{k=1}^{K=10}) \\ \frac{dy_4}{dt} = f(\{y_j\}_{j=1}^{N=7}, \tau_4, \{w_{14,j}\}_{j=1}^{N=17}, \theta_{14}, \{I_k^D\}_{k=1}^{K=10}) \\ \frac{dy_5}{dt} = f(\{y_j\}_{j=11}^{N=7}, \tau_5, \{w_{15,j}\}_{j=1}^{N=17}, \theta_{15}, \{I_k^D\}_{k=1}^{K=10}) \\ \frac{dy_6}{dt} = f(\{y_j\}_{j=1}^{N=7}, \tau_6, \{w_{16,j}\}_{j=1}^{N=7}, \theta_{16}, \{I_h^P\}_{h=1}^{H=2}) \\ \frac{dy_7}{dt} = f(\{y_j\}_{j=1}^{N=7}, \tau_7, \{w_{17,j}\}_{j=1}^{N=7}, \theta_{17}, \{I_h^P\}_{h=1}^{H=2}) \end{cases} \quad (3.1)$$

The nodes y_4 and y_5 were chosen to be the output neurons directly connected with the effectors. Total number of parameters are shown in Table 3.1:

These equations identify a family of dynamical systems, among which the right values suitable for performing the task have to be chosen.

Second layer. the second layer, L_2, is composed of a single self-connected neuron in such a way as to have two stable equilibrium points p_1 and p_2 (see the Section 5.1.2). The output of L_2 is the $\mathbf{I}^P$ input for Layer L_1. The output of the L_1 neuron controlling ω is given as $\mathbf{I}^D$to the L_2 neuron.

In order to set the $\mathbf{W}$ matrix of the above CTRNN, we have *trained* the layer to implement behaviourLeft() when $\mathbf{I}^P < 0$ and behaviourRight() when $\mathbf{I}^P > 0$. This is a machine learning task for which we decided to use an evolutionary approach starting from (Floreano and Mondada, 1994). We traced the v values of the robot sensors for a total number of $V = 1000$ during the robot execution; we construct for each controller a fitness with three elements:

- a contribute which grows with the distance which of the robot have from walls (measured by the sonar values S_k^v, $k \in \{1, \ldots, 10\}$);
- a contribute which grows with robot linear velocity (linear velocity values V_L^k);
- a contribute which grows inversely with respect to the angular velocity

The goal is to have a controller capable of avoiding walls, being fast and turning only when necessary. The cost function (fitness) assumes the form

$$F_{FM} = (1 - S_{max}) \cdot \bar{V}_L \cdot \left(1 - \sqrt{\bar{V}_A}\right)$$

with

$$\begin{aligned} S_{max} &= max\left\{S_k^v : k \in 1, ..., K; i \in 1, ..., 10\right\} \\ \bar{V}_L &= \frac{1}{K}\sum_{k=1}^{K} V_L^k \\ \bar{V}_A &= \frac{1}{K}\sum_{k=1}^{K} \left|V_A^k\right| \end{aligned}$$

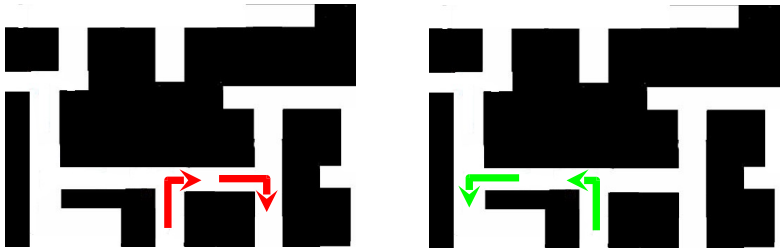

Figure 3.2: BehaviourRight() and BehaviourLeft() in a simulated T-maze environment in the software *Player/Stage/Gazebo*. As it is possible to see, BehaviourRight() corresponds to a right-wall-follower, and BehaviourLeft() to a left-wall-follower.

The second layer L_2 has been built so as to select p_2 when $\omega < \Omega_1$ and p_1 when $\omega > \Omega_2$, where $\Omega_1 < \Omega_2$ are fixed thresholds. In other words the system runs program behaviourLeft() (behaviourRight()), when it "realizes" that the robot has turned right (left). This system was tested in the environment simulator *Player/Stage/Gazebo* (Koenig and Howard, 2004). $\mathbf{I}^D$ is updated about every $10^2\,ms$ (the time scale T^F); Layer L_1 converges to a good approximation of the stable equilibrium point in about $10\,ms$ (the time scale T); the time between the switching of the programs is longer than $10^3 - 10^4\,ms$ (the time scale T^S). The entire system obtained by the composition of the two layers L_1 and L_2 succeeded in controlling the robot inside various *different size* multiple $T-$maze environments.

Analogously to works cited in the previous section two kinds of high level distinct behaviours were obtained on a fixed architecture by merging two layers. The lower layer performs the two programs by varying the auxiliary/context/program

Figure 3.3: Output of the CTRNN controller together with switching input versus integration step. We see after each turn in the $T-$maze layer L_2 force the behaviour to be performed by L_1.

input, while the data inputs are the sensory data on a slower time scale. However this approach gives rise to several questions to which without a clear model of the phenomenon it is not possible to answer. How many programs can we insert into a fixed structure architecture like the one just described? How can we insert new programs without deleting old ones? How can we even talk about programs if we do not know the effect of varying the values of what we call programs?

Notice that those questions could be answered by an architecture satisfying the properties a., b. and c. of Chapter 1 which resemble the typical programmability capabilities of a computational system. Thus our strategy will be to provide our CTRNN system with some features capturing programmability in a broader way, inside CTRNN framework. In order to do this we will build systems endowed with the *virtuality* capability.

3.3 Virtuality and programmability

What is virtuality? By virtuality in computational systems we understand the well-known capability of interpreting, transforming, and running machines which are present, as it were, only under their code (program) aspect. We deem a class of symbol-processing systems, both artificial and biological, to possess virtuality (Garzillo and Trautteur, 2009) if:

1. there exists an effective encoding of the structure of the single systems;

2. the codes provided by such encoding can be applied to specific systems of the class, designated as universal (or

interpreters), so that the universal system realizes the behaviour of the coded ones;

3. the codes can be processed by the systems of the class on a par with the input and output variables.

While the above definition arises within the Theory of effective computability, the specific use of the term 'virtuality' was introduced in the context of actual computing practice as in "virtual memory" or "virtual machine" and conveys an explicit attention to the discrimination between hardware and software. Thus a virtual machine is a code which behaves on a given, *fixed* material substrate as a different physical machine. Virtuality, therefore, exempts from the construction of different material structures in order to obtain different functions or behaviours, substituting construction with a description, namely the code. Furthermore Clause (2), allowing the processing of the codes as running I/O, grants the capability of virtually constructing or modifying other systems of the class and the modification of a system by itself.

We maintain that the ANN model proposed in this paper possesses the properties expounded in a., b., and c. above in Chapter 1. Property a. is discussed in Chapter 2; in Chapter 4 property c. will be shown to hold because for any N we will exhibit a CTRNN with a polynomially related number of neurons universal for all N-neurons CTRNN, while the all important property b. holds because the ANN model possesses virtuality. This virtuality is achieved by "pulling out" from the CTRNN the multiplication operation, commonly used to model the input of biological neurons, by introducing, for each connection, a subnetwork providing the outcome of the multiplication operation between the output coming from the pre-

synaptic neuron and the weight associated with the connection itself. As a consequence the weights can be given as auxiliary inputs to the original network augmented with the multiplication subnetworks, thus creating a neural architecture with two kinds of input lines: auxiliary (or programming) input lines and data input lines. Therefore, the auxiliary input lines can be fed with a code describing the network structure in a way resembling the implementation of a virtual machine in standard computational practice or the Gödel numbers fed into a Universal Turing machine.

It is easily checked that our proposed architecture satisfies

- Clause (1) – the connectivity pattern is the code
- Clause (2) – the multiplication subnetworks enable the behaviour of the coded network
- Clause (3) – the weights are the values of the auxiliary inputs

thus possessing virtuality as will be later stressed in Chapter 6.

Flexibility is the most prominent behavioural consequence of virtuality and is the requested capability we look for in neural systems. An entity with virtuality is flexible in the sense that it may exhibit a variety of different shapes of behaviour in response to the same sets of environmental data input patterns, depending on the program it is executing on its fixed basic structure.

In theoretical parlance, as distinguished from computational practice, virtuality is equated with universality in the sense that a single system – a Turing machine, deemed universal – is able to perform through simulation all computations per-

formed by any Turing machine: hence the term 'universality', which refers to the universe of the partial recursive, or partially computable, functions.

Our interest in virtuality does not stem from universality, but from programmability which allows the flexibility feature mentioned above. In particular in the chosen domain of CTRNNs presented in Chapter 2 we do not look for a universal CTRNN which might simulate the behaviour of any CTRNN, but for programmable CTRNNs which under different programs can exhibit any of an appropriate range of shapes of behaviour. A further step will be the implementation, always as a pointer to neurobiological research, of (re)programming parts of a NN. As the "programs" in CTRNNs will be, as we will see in Chapter 4, specifications of the weights of the NN, and these will be "pulled out" from a structural specification into *auxiliary* input variables, it is perfectly conceivable that parts of the NN might provide, with their output variables, the auxiliary or programming inputs for some other, or the same, parts of the total NN. This is the reason for including Clause (2) explicitly in our definition of virtuality.

3.3.1 Virtuality in biological systems

Natural, i.e. biological, symbolic systems have been functionally modelled by computational means, both with regard to lower level animals (see, for instance Lambrinos et al., 2000; Reeve et al., 2005) and mammals and humans as attested by a wealth of successful computational accounts of mental performances of a perceptive, cognitive and, recently, also emotional nature.

As is well known, such an approach has been the mainstay

of Artificial Intelligence and the associated synthetic method, where the actual understanding of the material behaviour of the nervous system has always been deferred, pending the discovery of the actual implementation of the computational process warranted by the multiple realizability hypothesis.

It now appears that the original notion of a biological nervous system actually implementing algorithmic processes has largely been superseded. Accordingly the dynamical systems approach to modelling biological control and symbolic behaviour has superseded Artificial Intelligence methodology. Yet the word 'computation' is still being used in a variety of often misleading or poorly understood ways. As examples we may consider the use of 'computational' within the "computational neuroscience" where 'computational' usually refers to some neural network (NN) used to model some part of the biological nervous system (Riesenhuber and Poggio, 2002), or in phrases such as "Biology is computational" (Fontana, 2006; Regev and Shapiro, 2002). Most of these usages are of a metaphorical nature.

In Chapter 1 high level biological tasks are presented, which seem to be clamoring to be interpreted as genuine computational tasks, and such as to need programming, despite the dynamical systems nature of the underlying, neural, basis. By 'programming' we explained that is not meant to design the NN for predetermined goals, but rather the fact that a fixed structure – functionally an *interpreter* – can be conditioned (programmed) by an auxiliary input so as to exhibit a repertoire of different shapes of behaviour.

This is the gist of *virtuality*, insofar as a single, fixed entity may be made to behave as a number of different "virtual", not materially present, entities.

These properties seem inexplicable, barring the existence of algorithmic performance in the nervous system. Indeed Sloman (Sloman and Chrisley, 2003; Sloman, 2008) has introduced the notion of virtual machine in the modelling of mental processes, albeit without inquiring about the material realizability of virtuality in the nervous system, while in other works (Donnarumma et al., 2007; Trautteur and Tamburrini, 2007; Garzillo and Trautteur, 2009) it has been investigated, or rather formulated, the question whether virtuality is actually present in biological brains.

The search for such programmable/virtual performance in artificial, but biologically plausible, neural architectures might be considered as the first step of a path of research aimed at detecting actual computing in the brain, as contrasted with the current trend striving at the identification of topographical areas associated with unique functionalities via EEG, PET, fMRI, etc. and at tracking interconnections between those areas, but largely neglecting the actual symbolic processing of nervous tissue.

The discovery of neural architectures, at first in artificial models, later in biological ones, supporting programming/virtuality, may open the way to an objective, and not merely metaphorical or functional, interpretation of neuronal activity as computational.

3.3.2 Turing Virtuality versus Material Virtuality

In the field of unconventional or alternative computing, where the goal of effective but not recursive symbol processing keeps being sought after, a number of systems not immediately computational in the algorithmic sense are shown to be capable of

(at least) simulating in some well defined sense a generic Turing machine, thus including a Universal Turing machine which is the theoretical underpinning of (Turing) virtuality.

The universality property and therefore virtuality is correctly assigned to those systems. In particular such is the case for a dynamical system simulation of algorithmic devices (Graça et al., 2005; Branicky, 1995), for a universal NN[1] implementing a Universal Turing machine (Siegelmann and Sontag, 1995; Siegelmann, 1999), for *liquid state machines* (Maass et al., 2002; Steil, 2004). This last system, can in particular be seen from the point of view of the topology of the network as the dual of our model. In fact in this work a fixed *universal kernel* or *reservoir* recurrent neural networks feeds another network in order to obtain a multi purpose system. In the present approach the multi purpose (programmable) network will be fed by a programming network (see Figure 3.4).

However, in neither of the three models, Clause (2) above is not fulfilled by the encodings provided in those papers. Indeed, in the simulation or implementations mentioned there appear to be two encodings: the explicit encoding of the Turing machine into the simulating system, and the never alluded-to encoding of a generic Turing machine, perhaps through certain Gödel numberings, on the tape of the simulated Universal Turing machine. The second and crucial encoding does not seem to be immediately accessible from the level of the simulating agencies - dynamical systems (Graça et al., 2005; Branicky, 1995), rational weights NN (Siegelmann and Sontag, 1995; Siegelmann, 1999) or universal resevoir NN (Maass et al., 2002; Steil, 2004)). In the present approach, in the spirit

[1]See also Appendix B for a detailed treatment.

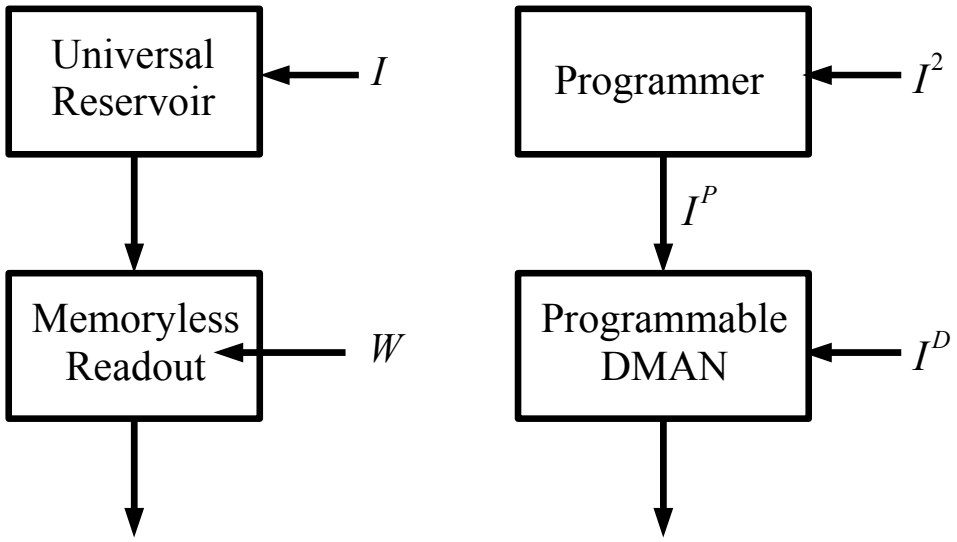

Figure 3.4: Comparing the topologies of a Liquid State Machine (on the left) and a DMA (on the right) which will be defined in Chapter 4. In Liquid State Machines the universal reservoir formed by a recurrent network feed an output subsidiary network. In DMA a Programmable Network possessing virtuality is fed by a programming network.

of biologically plausible modelling, we do not simulate other computational systems (Turing machines). We search for virtuality within the model itself, which we might call CTRNN or *material* virtuality, in the hope that this capability might be transferred, or rather discovered, in biological reality.

Accordingly, in this work we show how in the dynamical systems of the NN variety, in particular in CTRNNs, the crucial programming feature – virtuality – is realizable and at the same time we propose and implement examples of a promising architecture: the *dynamical multiplication architecture*.

4

Dynamical Multiplication Architecture

In this Chapter we introduce the core part of the thesis, the underlying principles and the detailed structure and organization of the Dynamic Multiplication Architecture. The approach will allow for the construction of interpreters, Dynamic Multiplication Architecture Networks (DMAN), possessing virtuality and thus, resulting programmable. The approach will be developed inside the CTRNN framework, so DMANs will turn out to be as interpreters of specific classes of CTRNNs.

4.1 Programmability through dynamical multiplication

The input to biological neurons is usually modelled as a sum of products between output signals coming from other connected neurons and the weights associated with the connections. So the evolution of a network is grounded into the sums of the products between weights and output signals. As indicated in Section 3.3, in the present approach the role of the weights is equivalent to programs in standard computational systems.

Here we propose to "pull out" the multiplication operation by using subnetworks providing the outcome of the multiplication operation between the output and the weight. As a consequence the weights are given as an auxiliary input to the original network augmented with the multiplication subnetworks, thus creating a neural architecture with two kinds of input lines: *auxiliary* (or *programming*) input lines in addition to standard *data* input lines. Notice that the newly introduced auxiliary input lines are fed with a code describing the original network in a way resembling the code of a virtual machine in a standard computational architecture or the Gödel numbers given to a Universal Turing machine.

4.2 DMA explained

Let us suppose we have an ideal CTRNN, mul, composed of M neurons some of which (input neurons) are fed with inputs $a, b \in (0, 1)$ with appropriate weights, and one is an output neuron k, with the steady state $\bar{y}_k$ so that $\sigma(\bar{y}_k) = a \cdot b$.

Given a simple network S composed of only two neurons i and j linked by just one connection with weight $w_{ij} \in (0, 1)$, it is possible to build an "equivalent" Dynamical Multiplication Architecture Network (DMAN) S_{mul} by means of the *multiplication* network mul according to the following steps (see Figure 4.1):

1. redirect the output of the neuron j as input a of mul
2. set the input b of mul to w_{ij}
3. redirect the output of the neuron k of mul as input to the neuron i with weight 1

In this case, supposing that the *mul* time scale of the approach to the stable point is so fast as to make completely negligible its computational delay with respect to the S time scale, the dynamic behaviour of the constructed DMAN S_{mul}, restricted to the neurons i and j, is equivalent (identical in case of zero delay of the *mul* subnetwork) to the original S.

It is always possible to extend this procedure to a generic weight $w_{ij} \in (min, max)$. We have the problem of creating a network which reproduces the product $w_{ij} \cdot \sigma(y_j)$ with $w_{ij} \in (min, max)$. We can rescale the parameter w_{ij} with the transformation

$$w_{ij} \cdot \sigma(y_j) = (max - min) \cdot p \cdot \sigma(y_j) + min \cdot \sigma(y_j)$$

with $p \in (0, 1)$.

This means that we can substitute the $w_{ij}-$connection with a connection with weight min plus a connection with weight $(max - min)$ coming from the multiplication network mul, which receives as inputs $\sigma(y_j)$ and the programming input $p = (w_{ij} - min)/(max - min)$ as shown in Fig. 4.2.

In the rest of the paper, each time such procedure will be applied, we will refer to it as *w-substitution*.

Now given a generic CTRNN G composed of N neurons, with inputs $\mathbf{x} = [x_{N+1}, \ldots, x_{N+L}]$, and weights $w_{ij} \in (min, max)$, with $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, N + L\}$, let us construct a DMAN G_{mul} applying the w-substitution uniformly.

In this way, a DMAN G_{mul} is obtained, composed of $N + N \cdot M \cdot (N + L)$ neurons[1], with data inputs $\mathbf{x} = [x_{N+1}, \ldots, x_{N+L}]$, and auxiliary inputs $\mathbf{p} = [p_1, ..., p_{N(N+L)}]$, which has the same

[1]Notice that the number of neurons in a $w-$substituted DMAN G_{mul} grows as $O(N^2)$, well within polynomial bounds, with respect to the number of neurons of G.

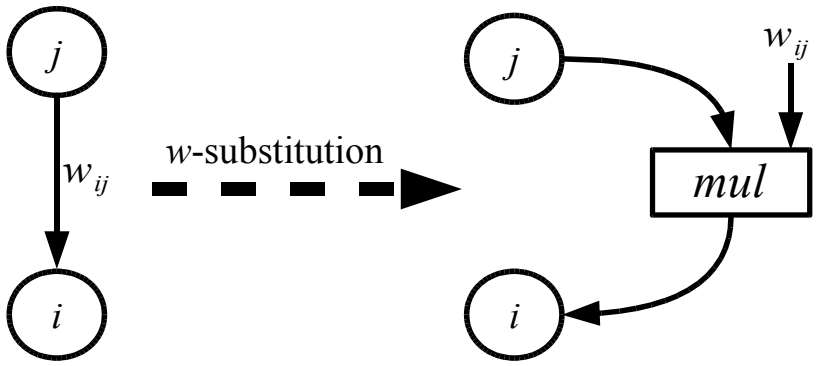

Figure 4.1: The w-substitution procedure for weights $w \in_{ij} [0, 1]$. For each weight a *mul* net, fed with the appropriate program $p = w_{ij}$, is inserted.

dynamic behaviour as G, if restricted to the neurons in G.

As a consequence the network G_{mul} fed with any auxiliary inputs $p_h \in (0, 1)$ has the same behaviour as the network G with weights[2],

$$w_{ij} = (max - min) \cdot p_h + min \tag{4.1}$$

if restricted to the neurons in G.

4.3 DMA properties

Hence, the DMAN G_{mul} represents an artificial system which possesses virtuality because it fulfills Clauses (1), (2) and (3) defined in Section 3.3. In fact, the p_h effectively codes an N neuron, L input CTRNN and gets decoded by Eqn (4.1). The

[2] i is equal to the result of the integer division $(h-1)/(N+L)$ plus 1, while j is equal to the remainder plus 1

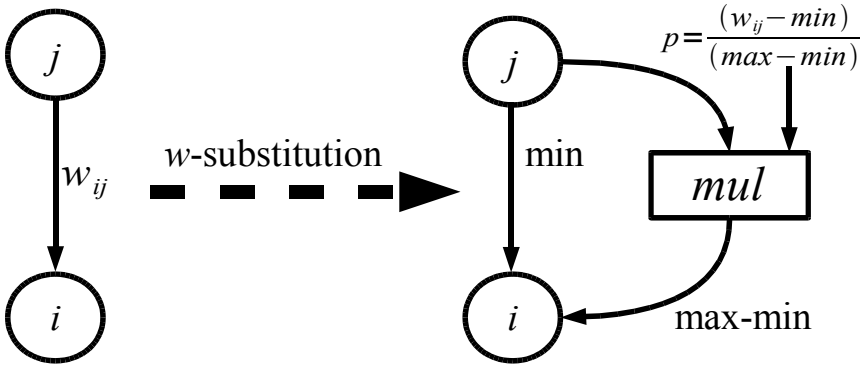

Figure 4.2: The w-substitution procedure for weights $w_{ij} \in [min, max]$. For each weight a mul net, fed with the appropriate program $p = (w_{ij} - min)/(max - min)$, is inserted .

code, p_h, can be given to G_{mul} which realizes the behaviour of the coded one. Also the code can be processed on a par with the data input variables.

So in this sense we will say that p_h auxiliary inputs play the role of a program which is able to determine the suitable behaviour of the G_{mul} as a network G with assigned weights.

Notice that a number of neurophysiologic findings suggests the presence of biological neurons showing a multiplicative response on some input signals[3]. For example, the multiplication is thought to play a crucial role in coordinate transformation (Andersen et al., 1997) or auditory processing (Pena and Konishi, 2004). The presence of neurons showing as response a multiplicative operation on some input signals could be pre-

[3]Recent research on the glia suggests the possibility of modulation of synaptic efficacy by astrocytes (Haydon and Carmignoto, 2006). If this were the case our proposal of searching for programmability in biological networks would be corroborated insofar as the glia might be interpreted as a "programming" unit for grey matter neurons. Here we do not further develop such hypothesis.

sumably explained in two ways (Salinas and Abbott, 1996):

- by the existence of single neurons which are able to perform a multiplication operation on their incoming signals (but biological neurons are usually modelled by a weighted sum);
- by the interaction of a population of neurons (a subnetwork), where just one output neuron (or few neurons) shows a multiplicative response, even if all the individual neurons sum their synaptic input linearly and are not able to perform a multiplication operation singularly.

Although a number of abstract models of neurons with multiplicative response on their synaptic inputs (known as $\Sigma - \Pi$ units) have been proposed in the field of artificial neural networks (Rumelhart et al., 1986; Younger et al., 1999), we consider the subnetwork explanation a corroboration of our theoretical introduction of the CTRNN *mul*.

5

A theory for comparing DMANs

In Chapter 4 we presented the DMA model which, starting from a network G and applying the $w-$substitution procedure, allows to construct a DMAN G_{mul}, which is an interpreter in the computational sense for specific classes of CTRNNs. However, as already stated in the previous chapter, the class that a DMAN captures is well defined only in the presence of subnetworks mul with an ideal attractor computing behaviour, which implies an exact approach to their fixed points in a zero time delay. In a real implentation, of course, this zero time delay cannot be assumed, and is also biologically implausible. Thus finite time delay, and not exact approach to the fixed points should be assumed, and real approximated implementation of mul, which we will call $mul^{\star}$ will be deployed. Of course the interpreting capability is reduced. In order to measure the residual interpreting capability and to demonstrate the feasibility of the approach even as far as the actual implementation is concerned, we need to compare the original G network behaviours, to the $w-$substituted networks $G_{mul^{\star}}$. As we hinted in Chapter 2, approaches based on the universal dynamical

approximation theorem are not helpful because that theorem compares one solution at a time of the dynamical systems involved. So in this section, starting from Bifurcation Theory, we introduced a number of very recent techniques for dynamical systems which take their inspiration from Formal Verification Methods and that can be summarized as *Bisimulation Techniques,* which will bring us a measure which enables us to compare CTRNN dynamical systems, and specifically the DMA network interpreters versus the original CTRNN they simulate. Notice that these techniques are a very new branch of research in dynamical system theory and this is the first attempt, to the best of our knowledge, to specialize bisimulation inside Neural Networks framework.

5.1 Background notions in Bifurcation Theory

In studying dynamical systems, we are not only interested in specific solutions of a specific system, but we want to classify dynamical systems according to their general qualitative shapes of behaviour. One first step in this direction is to compare the number, position and stability of their invariant sets. This aspect will become especially important in the context of the DMA when searching for *interpreters* of classes capable of simulating classes of CTRNNs.

5.1.1 Topological Equivalence

Definition 5.1.1. Two dynamical systems $D_1 = (X_1, \gamma_1, T_1)$ and $D_2 = (X_2, \gamma_2, T_2)$ are *topologically equivalent* if there exist a homeomorphism $h : X_1 \longrightarrow X_2$ mapping orbits of the first system on the second system.

Definition 5.1.2. Two dynamical systems $D_1 = (X_1, \gamma_1, T_1)$ and $D_2 = (X_2, \gamma_2, T_2)$ are *topologically conjugate* if they are topologically equivalent and the trajectories evolve with the same speed.

Definition 5.1.3. Two dynamical systems $D_1 = (X_1, \gamma_1, T_1)$ and $D_2 = (X_2, \gamma_2, T_2)$ are *topologically C^k equivalent* if there exists a diffeomorphism $h : X_1 \longrightarrow X_2$ mapping orbits of the first system on the second system.

Definition 5.1.4. A a family of dynamical systems $D = (X, \gamma^r, T)$ depending on a parameter k is locally *structurally stable* in $r = \bar{r}$ if any perturbation of $\gamma^{\bar{r}+\epsilon}$ near to $\bar{k}$ is topologically conjugate to $\gamma^{\bar{r}}$.

Theorem 5.1.5. *Each additive model of continuous time recurrent neural network is topologically conjugate to a sigmoid CTRNN. Moreover given the equation*

$$\boldsymbol{\tau}\frac{d\mathbf{y}}{dt} = -\mathbf{y} + \mathbf{W}\boldsymbol{\sigma}\left(\mathbf{y} - \boldsymbol{\theta}\right) + \mathbf{I}$$

is topologically equivalent to

$$\boldsymbol{\tau}'\frac{d\mathbf{y}'}{dt} = -\mathbf{y}' + \mathbf{W}'\boldsymbol{\sigma}_{a,b,c}\left(\mathbf{y}' - \boldsymbol{\theta}'\right) + \mathbf{I}'$$

given by the homeomorphism

$$\begin{aligned} \mathbf{y}' &= c^{-1}\mathbf{y} \\ \boldsymbol{\tau}' &= \boldsymbol{\tau} \\ \mathbf{W}' &= (ac)^{-1}\mathbf{W} \\ \boldsymbol{\theta}' &= c^{-1}\boldsymbol{\theta} \\ \mathbf{I}' &= c^{-1}\mathbf{I} + abc\mathbf{W} \cdot \mathbf{1}_N \end{aligned}$$

where $\mathbf{1}_N = [1, \ldots, 1]$ *is the identity row vector of dimension* N.

This theorem, proved in (Haschke, 2004; Beer, 2006), shows that all the results obtained in this thesis for the DMA can be easily transferred to any of the architectures with activation functions $\sigma_{a,b,c}(x)$ since for each sigmoid CTRNN found it is possible to construct an equivalent $\sigma_{a,b,c}$ CTRNN.

Theorem 5.1.6. *(Grobman - Hartman) If* $\bar{\mathbf{x}}$ *is a hyperbolic equilibrium point of a dynamical system* $D = (X, \gamma, T)$, (2.3) *then there is a neighbourhood of* $\bar{\mathbf{x}}$ *in which* D *is topologically conjugate to the linear system* $\dot{\bar{\mathbf{x}}} = J(\bar{\mathbf{x}})\mathbf{x}$.

Thus Bifurcations points are values of a parametric dynamical system $\gamma^r(t, x)$ for which small variation of r alters the structure of its surface stability. In the next section analyzing the CTRNN neuron we will find two kind of bifurcation points: *Saddle Node* bifurcations and *Pitchfork* bifurcations. Figures 5.1 and 5.2 illustrate the typical bifurcation diagram of these bifurcations. A bifurcation diagram plots the xixed points as function of the parameter r of the system. As we can see, a system which undergoes a saddle node bifurcation detroys its two fixed points, one stable and one unstable, which annihilate

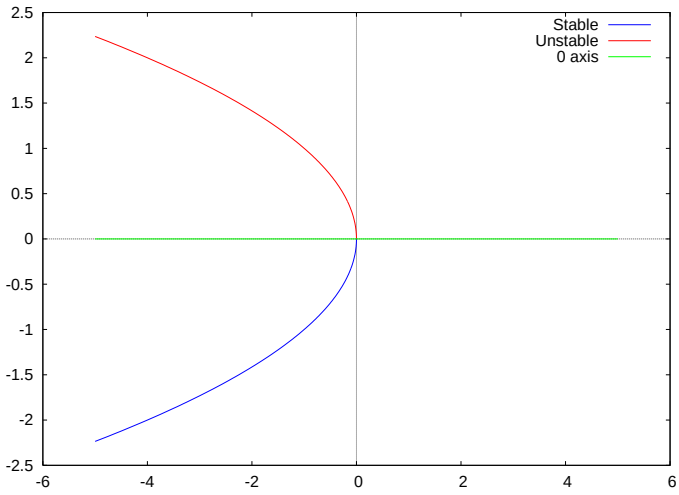

Figure 5.1: Bifurcation Diagram for a saddle node bifurcation. red lines are unstable fixed points, blue lines are stable fixed points.

or disappear; on the other hand a system which undergoes a pitchfork bifurcation split its stable fixed point into three fixed points: one unstable and two unstable.

5.1.2 Analysis of simple networks: one neuron analysis

The understanding of the dynamical behaviour of a CTRNN system is a difficult task as much as the number of neurons constituting the system increases. In Chapter 6 we started from testing a DMAN in *simulating* the dynamic behaviour of only one neuron. In this section we analyze the dynamics of a CTRNN consisting of only one neuron $NetOne$, the behaviour of which will be compared with the DMAN interpreter $NetOne_{mul^\star}$ simulating only one neuron.

Thus let us consider the case of a CTRNN equation written up for a single neuron with a self-connection. Equation 2.1

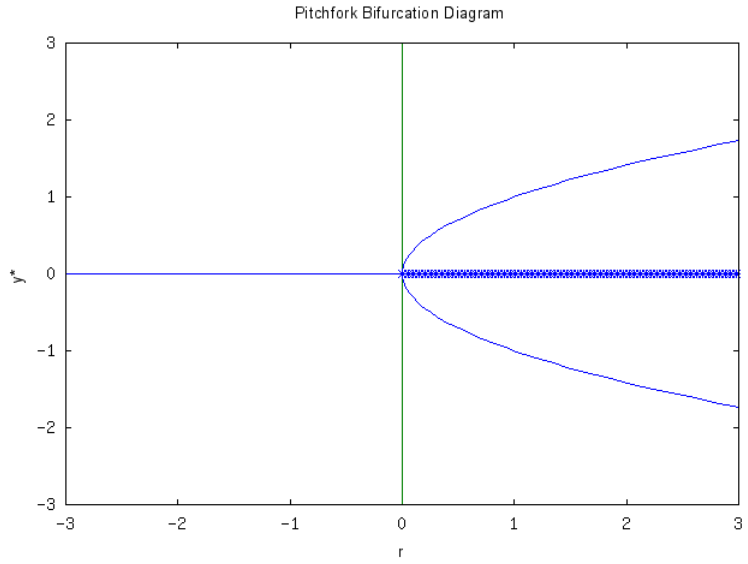

Figure 5.2: Bifurcation diagram for a Pitchfork Bifurcation. The continuous blue lines are stable fixed points, the dotted blue lines are unstable fixed points.

reduces to

$$\dot{y} = -y + w\sigma\left(y - \theta\right) + I \tag{5.1}$$

where for simplicity we set the time constant $\tau = 1$. Notice that no elementary expression for the solution of (5.1) exists. By contrast, it is possible to achieve a complete qualitative description of its dynamics (Beer, 1995b). Specifically, one can describe the limit sets of (5.1), including their stability and their dependence on the parameters, as well as the bifurcations that can occur as the parameters are varied (see Fig. 5.3). Such system has a cusp point (Hale and Koçac, 1991). In this system the cusp point $\left(\tilde{I}, \tilde{w}\right)$ is the only bifurcation point in which the system undergoes a pitchfork bifurcations (Hale and Koçac, 1991). All other bifurcation points are saddle-node bifurcations.

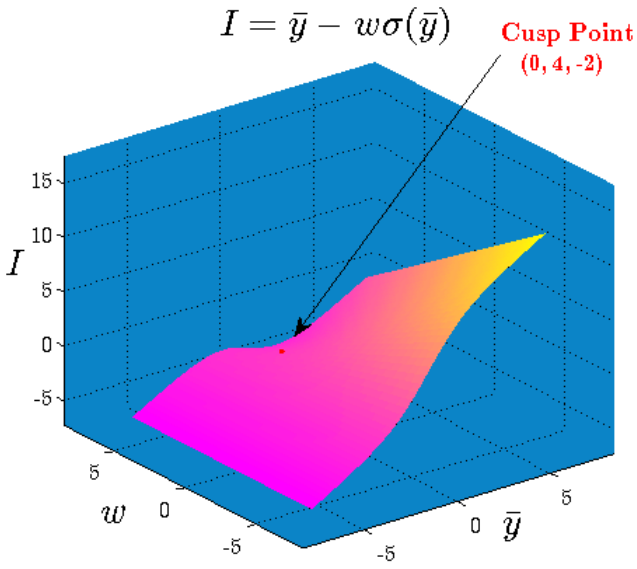

Figure 5.3: Stability Surface and Cusp point of *NetOne*

To find the bifurcation point, we recall that Theorem 5.1.6 points out that bifurcation points are *non-hyperbolic* equilibria $\bar{y}$; thus from Chapter 2 we find the conditions:

$$f(\bar{y}) = 0 \tag{5.2}$$

$$f'(\bar{y}) = 0 \tag{5.3}$$

Computing these conditions to 5.1

1. $\bar{y}$ must be a fixed point (condition 5.2):

$$-\bar{y} + w\sigma\left(\bar{y} - \theta\right) + I = 0 \tag{5.4}$$

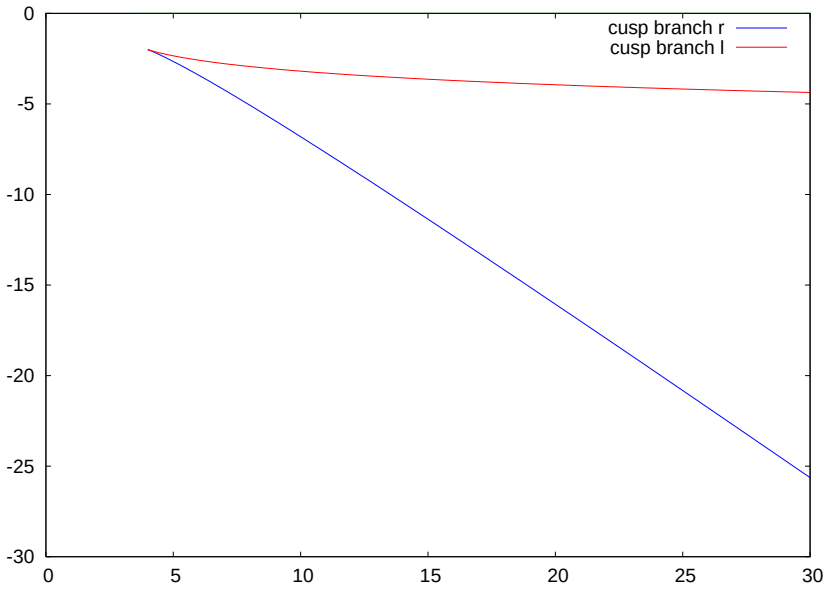

Figure 5.4: The two branches of the cusp for *NetOne* with $\theta = 0$

2. $y - I$ and $w\sigma(y)$ have to be tangent in $\bar{y}$:

$$1 = w\sigma'(\bar{y} - \theta) = w\sigma(\bar{y} - \theta)(1 - \sigma(\bar{y} - \theta)) \tag{5.5}$$

There are two solutions for this equation: in fact studying the condition (5.5), we obtain

$$\frac{1}{w} = \sigma(\bar{y} - \theta)(1 - \sigma(\bar{y} - \theta)) = \frac{e^{-(\bar{y}-\theta)}}{\left(1 + e^{-(\bar{y}-\theta)}\right)^2}$$

and setting

$$z = e^{-(\bar{y}-\theta)} \tag{5.6}$$

we have

$$wz = 1 + z^2 + 2z \Longrightarrow 1 + z^2 + (2 - w)z = 0$$

with the two solutions

$$z = \begin{cases} \frac{w-2+\sqrt{(w-2)^2-4}}{2} \\ \frac{w-2-\sqrt{(w-2)^2-4}}{2} \end{cases}$$

Remembering (5.6) we can write

$$\bar{y} = \begin{cases} f_1(w,\theta) \equiv -\ln\left(\frac{w-2+\sqrt{(w-2)^2-4}}{2}\right) - \theta \\ f_2(w,\theta) \equiv -\ln\left(\frac{w-2-\sqrt{(w-2)^2-4}}{2}\right) - \theta \end{cases}$$

We have found the expression for a fixed point $\bar{y}$ in function of the weight w. If we substitute them in (5.4) we obtain the two curves

$$I(w,\theta) = \begin{cases} f_1(w,\theta) - w\sigma(f_1(w,\theta) - \theta) \\ f_2(w,\theta) - w\sigma(f_2(w,\theta) - \theta) \end{cases}$$

which are those of the cusp (see Fig. 5.4). These curves intersect only in one point, the cusp point, when the two solutions are identical, $I_1 = I_2$

$$\begin{aligned} &-\ln\left(\frac{w-2+\sqrt{(w-2)^2-4}}{2}\right) - \theta - w\sigma\left(-\ln\left(\frac{w-2+\sqrt{(w-2)^2-4}}{2}\right) - 2\theta\right) = \\ &-\ln\left(\frac{w-2-\sqrt{(w-2)^2-4}}{2}\right) - \theta - w\sigma\left(-\ln\left(\frac{w-2-\sqrt{(w-2)^2-4}}{2}\right) - 2\theta\right) \end{aligned} \tag{5.7}$$

Fixing θ we can found solution for the solution for the condition (5.7). Taking, for example, the case in which $\theta = 0$, the logarithm arguments on the left and on the right have to be

equal, so

$$\frac{w-2+\sqrt{(w-2)^2-4}}{2} = \frac{w-2-\sqrt{(w-2)^2-4}}{2} \Longrightarrow$$
$$(w-2)^2 - 4 = 0 \Longrightarrow$$
$$w^2 - 4w = 0$$

One solution is $w = 4$ ($w = 0$ is not good because it makes the logarithm argument inferior to 0). If we substitute $w = 4$ in I we obtain $I = -2$. So the coordinates of the cusp point are $\left(\tilde{I}, \tilde{w}\right) = (-2, 4)$.

As far as this analysis show, even the simplest element of a CTRNN is a complex dynamical system. In Chapter 6 a DMAN interpreter of this complex dynamical simulation, $NetOne_{mul^\star}$, will be constructed and analyzed.

5.2 A formal definition of abstraction

Theorem 5.1.6 gives us a powerful method in order to analyze classes of dynamical systems the stability surface of which is composed only by fixed points, by comparing their behaviour in proximity of their fixed points. Thus for two systems D_1 and D_2 with the same number of fixed points, none of them being non-hyperbolic, we can in theory perform an "equivalence measure" by:

- computing the eigenvalues λ of the Jacobian for each fixed point of the systems
- computing for each eigenvalue the multiplicity n_- of the eigenvalues with real part inferior to zero and of eigenvalues n_+ of the eigenvalues of real part superior to zero.

- then comparing if for each fixed point $\bar{\mathbf{x}}^1$ in D_1 there is one $\bar{\mathbf{x}}^2$ in D_2 $n^1_- = n^2_-$ and $n^1_+ = n^2_+$

Thus if the multiplicity is the same the systems have the same stability surface and they result to be locally topologically conjugate.

However all the approach is based on the assumption that it is possible to compute the fixed point of nonlinear CTRNN systems. Unfortunately determining the exact region of attraction analytically might be difficult or even impossible for a nonlinear dynamic system (Khalil, 2002). Thus in general topological equivalence is too strong a condition to be deployed in DMANs.

In the subsequent section, starting from Bisimulation definition, which rises from Formal Verification Theory (Clarke et al., 2000), and which is the homologous definition of topological equivalence for transition systems, we are able to relax the equivalency definition and to find a way to systematically compare CTRNN systems and so the interpreting capability of DMANs.

5.2.1 Transition systems associated with dynamical systems

Definition 5.2.1. A *transition system* $T \equiv (Q, \Sigma, \rightarrow, Q_0)$ consists of

- A set Q of states
- An alphabet Σ of events
- A transition relation $\rightarrow \subseteq Q \times \Sigma \times Q$
- A set $Q_0 \subseteq Q$ of initial states

Definition 5.2.2. A *bisimulation* between two transition systems $G \equiv (Q, \Sigma, \rightarrow, Q_0)$ and $G' \equiv (Q', \Sigma' \equiv \Sigma, \rightarrow', Q'_0)$ is an equivalence relation $\sim \subseteq Q \times Q^{'}$ such that,

- $\forall q_1, q_2 \in Q$, $\forall q'_1 \in Q' \forall a \in \Sigma (q_1 \sim q_1^{'}$ and $q_1 \xrightarrow{a}_{\gamma} q_2) \implies (\exists q_2^{'} \mid q_2 \sim q_2^{'}$ and $q_1^{'} \xrightarrow{a}{}' q_2^{'})$
- $\forall q'_1, q'_2 \in Q$', $\forall q_1 \in Q \forall a \in \Sigma$ $(q_1 \sim q_1^{'}$ and $q_1^{'} \xrightarrow{a}{}' q_2^{'}) \implies (\exists q_2 \mid q_2 \sim q_2^{'}$ and $q_1 \xrightarrow{a} q_2)$

The notion of Bisimulation states when two Transition systems are equivalent. Bisimulation is formally an equivalence relation and subsumes a partition of the states of the transition systems (Zhang, 1994).

On the other hand, looking at the general definition of dynamical system 2.6.1, it is possible to associate in a general way a transition system with them, basing on the flow γ of it (Brihaye, 2006).

Definition 5.2.3. The (labelled) transition system $G_\gamma \equiv (Q, \Sigma, \rightarrow_\gamma , Q_0)$ associated with a dynamical systems $D \equiv (X, \gamma, T)$ is defined by the following:

- the set Q of states is Y;
- the set $Q_0 = \{\gamma(\mathbf{x}_0, 0) \in Y \ \mathbf{x}_0 \in X_0\}$ with $X_0 \subseteq X$ the set of the initial conditions;
- the set Σ of events is T;
- the transition relation $y_1 \xrightarrow{\tau}_\gamma y_2 \subseteq Y \times T \times Y$ is given by $\exists \mathbf{x} \in X, \exists t_1, t_2 \in T$, $(t_2 - t_1 = \tau \wedge \gamma(\mathbf{x}, t_1) = y_1 \wedge \gamma(\mathbf{x}, t_2) = y_2)$.

Definition 5.2.4. The time abstract transition system $G_\gamma^A \equiv (Q, \Sigma, \rightarrow_\gamma, Q_0)$ associated to a dynamical systems $D \equiv (X, \gamma, T)$ is defined by the following:

- the set Q of states is Y;
- the set $Q_0 = \{\gamma(\mathbf{x}_0, 0) \in Y \, \mathbf{x}_0 \in X_0\}$ with $X_0 \subseteq X$ the set of the initial conditions;
- the transition relation $y_1 \rightarrow_\gamma y_2 \subseteq Y \times Y$ is given by
  $\exists \mathbf{x} \in X, \exists t_1, t_2 \in T, (t_1 \leq t_2 \wedge \gamma(\mathbf{x}, t_1) = y_1 \wedge \gamma(\mathbf{x}, t_2) = y_2)$.

Definition 5.2.5. A (time abstract) bisimulation on a dynamical system (X, γ, T) is an equivalence relation $\sim$ on Y such that the following property holds

$\forall y_1, y_1', y_2 \in Y$

$(y_1 \sim y_2) \wedge (y_1 \xrightarrow{\tau} y_1') \Rightarrow (\exists \tau' \in T, \exists y_2' \in Y, (y_1' \sim y_2') \wedge (y_2 \xrightarrow{\tau'} y_2'))$

Time abstract bisimulation gives partitions of the phase space of the dynamical partition. There always exists a trivial bisimulation on a dynamical system given by $\{Y\}$. However, non-trivial bisimulations give rise to the definition of the interesting concept of *abstraction* for the dynamical systems.

5.2.2 Semantics of a continuous system

Building on the definition of the previous section it is possible to associate a language with a continuous dynamical system.

Definition 5.2.6. A word ω_{x_0} on a partition $\mathcal{P}$ of the set Y from a dynamical system $D = (X, \gamma, T)$ is the succession of sets of the partition $\mathcal{P}$

$$\omega_{\mathbf{x}_0} \, : \, \mathcal{F}_{\mathbf{x}_0} \longrightarrow \mathcal{P}$$

where $\mathcal{F}_{\mathbf{x}_0}$ is a succession of intervals or points of T, determined by the trajectory $\gamma(t, \mathbf{x}_0)$, of the induced partition $\mathcal{F}$ on T constructed as $\{t \in T \mid \gamma(t, \mathbf{x}_0) \in \mathcal{P}\}$.

Definition 5.2.7. We denote by $\Omega_{\mathcal{P}}$ the set of words associated with the dynamical system (X, γ, T) with respect to a partition $\mathcal{P}$

The set $\Omega_{\mathcal{P}}$ gives a complete static description of the dynamical system

Definition 5.2.8. Given the set of intervals

$$\mathcal{F}_{(\mathbf{x},t)} = \{I \in \mathcal{F}_{\mathbf{x}} \mid I \geq I_t\}$$

the *suffix* of the world $\omega_{\mathbf{x}}$ associated with time t is the restriction

$$\omega_{(\mathbf{x},t)} = \omega_{\mathbf{x}}|_{\mathcal{F}_{(\mathbf{x},t)}}$$

Definition 5.2.9. The *suffix dynamical type* of $y \in Y$ with respect to a partition $\mathcal{P}$ of V_2, given a dynamical system (X, γ, T), is defined by

$$\mathrm{Suf}_{\mathcal{P}}(y) = \{\omega_{(\mathbf{x},t)} \mid \gamma(t, \mathbf{x}) = y\}$$

Definition 5.2.10. The *suffix partition* with respect to a partition $\mathcal{P}$ of a dynamical system (D, γ, T) is the partition induced by the equivalence relation on the phase space Y between points *having the same suffix dynamical type.*

This approach describes how trajectories of a given dynamical system D can be encoded through words on a given partition associated with D.

This word encoding technique can be used to build a new symbolic "procedure" for computing bisimulations. Given a dynamical system D and a partition $\mathcal{P}$ of the phase space, it is possible to build $Suf(\mathcal{P})$. The Bisimulation algorithm shown in Appendix C ensures that either $\mathcal{P} \equiv Suf(\mathcal{P})$ or $Suf(\mathcal{P})$

refines $\mathcal{P}$. In the former case we obtain the bisimulation $\mathcal{P}$. In the latter we iterate the algorithm until $Suf^i(\mathcal{P})$.

Lemma 5.2.11. *(Brihaye, 2006) Given a dynamical system D and a partition $\mathcal{P}$ of its phase space iterating the partition induced by Suf we obtain*

$$\mathcal{P} \prec Suf(\mathcal{P}) \prec Suf^2(\mathcal{P}) \prec \cdots \prec Suf^k(\mathcal{P}) \prec \ldots$$

In cases we know this procedure stops, we could utilized it to obtain a complete static description of the dynamical system (see Appendix C).

In general the possibility to exactly replace the dynamical systems with one which lives on a lower dimensional space is a practice going under the name of *reduction* of a dynamical system (see e.g. Antoulas et al., 2001).

The approach developed in the next section falls in some degree within this branch of techniques although it does not give an exact replacement of the starting system, but only an approximated one (see Tabuada et al., 2008).

5.3 Similarity measures

Exact bisimulations between two labelled transition systems require that their observations are (and remain) identical as stated in Definition 5.2.2). However, there are very few cases in which an exact abstraction of a non linear dynamical system can be performed.

The *Approximate* bisimulation approach presented in this section (see Girard and Pappas, 2005) is less rigid since it only requires that the observations of both systems are (and remain)

arbitrarily close.

We firstly considered extensions of the labelled transition systems including an *observation space* H and an *observation map* h.

Definition 5.3.1. A *Transition System with observables* $T \equiv (Q, \Sigma, \rightarrow, Q_0, H, h)$, *where*

- $(Q, \Sigma, \rightarrow, Q_0)$ *have the same meaning of definition 5.2.1*
- H *is the observation space, a metric space equipped with a metric* d
- $h : Q \rightarrow H$ *is an observation map which maps variables of the system on the observation space*

Again we can associate to each dynamical system $D = (X, \gamma, T)$ a Transition system with observables T , selecting a subspace of X and a transfer function h on which to compare systems. If h is simply a projection of a number of variables of X we talk about Transition systems with clean observables.

Definition 5.3.2. A *Transition system with clean observables* is a transition system with observables T in which the observation map $h : Q \rightarrow H$ being a projection.

In the special case in which we select the entire space X as observation space together the identity map as h we have a full observable Transition System.

Definition 5.3.3. A *Completely Observable Transition System* is a Transition system with observables $T \equiv (Q, \Sigma, \rightarrow, Q_0, H, h)$ in which $H = Q$ and h is the identity function.

Definition 5.3.4. A relation $\sim_\delta$ is a $\delta-$*approximate bisimulation* between the transition systems with observables $T_1 \equiv$

$(Q_1, \Sigma, \rightarrow_1, Q_0^1, H, h_1)$ and $T_2 \equiv (Q_1, \Sigma, \rightarrow_1, Q_0^1, H, h_1)$ with common labels Σ, and a common observation space H, if for all $(q_1, q_2) \in Q_1 \times Q_2$

1. $d(h_1(q_1), h_2(q_2)) \leq \delta$
2. $(q_1, q_2) \in \sim_\delta \wedge q_1 \rightarrow_1 q_1' \in Q_1 \Rightarrow \exists q_2' \in Q_2 \ (q_1', q_2') \in \sim_\delta \wedge q_2 \rightarrow_2 q_2'$
3. $(q_1, q_2) \in \sim_\delta \wedge q_2 \rightarrow_2 q_2' \in Q_2 \Rightarrow \exists q_1' \in Q_1 \ (q_1', q_2') \in \sim_\delta \wedge q_1 \rightarrow_1 q_1'$

Note 5.3.5. If $\delta = 0$ the definition in 5.3.4 collapses to exact bisimulation Definition 5.2.2.

Definition 5.3.6. The transition systems T_1 and T_2 are said to be *approximately bisimilar* with approximation δ ($T_1 \sim_\delta T_2$) if there exists a $\sim_\delta$, δ-approximate bisimulation, such that given the initial conditions Q_1^0, and Q_2^0

- $\forall q_1 \in Q_1^0 \ \exists q_2 \in Q_2^0$ such that $(q_1, q_2) \in \sim_\delta$
- $\forall q_2 \in Q_2^0 \ \exists q_1 \in Q_1^0$ such that $(q_1, q_2) \in \sim_\delta$

The $\delta-$approximate bisimulation between two transition systems *guarantees* that the distances between their language is bounded.

Theorem 5.3.7. *(Girard and Pappas, 2007)If two transition systems T_1 and T_2 are approximately bisimilar with approximation δ, then for all observable trajectories of T_1 $h_1(q_1^0) \stackrel{t_1}{\rightarrow} h_1(q_1^1) \stackrel{t_2}{\rightarrow}$ there exists a trajectory $h_2(q_2^0) \stackrel{t_1}{\rightarrow} h_2(q_2^1) \stackrel{t_2}{\rightarrow} \ldots$, such that $\forall i$, $d(h_1(q_1^i), h_2(q_2^i)) \leq \delta$ and viceversa.*

In the light of this definition it is possible to reformulate the Funahashi - Nakamura Theorem 2.4.1

Theorem 5.3.8. *(Funahashi - Nakamura reformulated) For every Autonomuos Continuous Dynamical System D and an initial condition $\mathbf{x}_0$, with its completely observable transition system $T \equiv (Q, \Sigma, \rightarrow_\gamma, \mathbf{x}_o, Q, id)$, $\forall \delta > 0$ $\exists N$ such that a CTRNN D^{CTRNN} with N networks with an associated Transition System with clean observables T^{CTRNN} such that T and T^{DTRNN} are δ-approximately bisimilar.*

The construction of approximate bisimulations between two transition systems as well as the evaluation of their precision can be performed using class of functions called *bisimulation functions,* which are positive functions defined on $Q_1 \times Q_2$, bounding the distance between the observations associated with a couple (q_1, q_2) and non-increasing under the (nondeterministic) dynamics of the systems.

Definition 5.3.9. A *bisimulation function* V_B is a continuous function

$$V_B : Q_1 \times Q_2 \rightarrow \mathbb{R}^+$$

with

1. $V_B(q_1, q_2) \geq d(h_1(q_1), h_2(q_2))$
2. $V_B(q_1, q_2) \geq \max_{q_1 \xrightarrow{t} q_1'} \min_{q_2 \xrightarrow{t} q_2'} V_B(q_1', q_2')$
3. $V_B(q_1, q_2) \geq \max_{q_2 \xrightarrow{t} q_2'} \min_{q_1 \xrightarrow{t} q_1'} V_B(q_1', q_2')$

Theorem 5.3.10. *(Girard and Pappas, 2007) If V_B is a bisimulation function, then $\forall \delta \geq 0$ the set*

$$B_\delta = \{(q_1, q_2) \in Q_1 \times Q_2, V_B(q_1, q_2) \leq \delta\}$$

is a δ-approximate bisimulation between T_1 and T_2.

Note 5.3.11. The zero set of a bisimulation function is an exact bisimulation between T_1 and T_2

Corollary 5.3.12. *If V_B is a bisimulation function between T_1 and T_2, if*

$$\delta = \max\{\max_{q_1 \in Q_1^0} \min_{q_2 \in Q_2^0} V_B(q_1, q_2), \max_{q_2 \in Q_2^0} \min_{q_1 \in Q_1^0} V_B(q_1, q_2)\} \quad (5.8)$$

then T_1 and T_2 are approximately bisimilar with precision δ

This is an important result which encompass the possibility of comparing family of solution of Dynamical systems, and so entirely Dynamical systems parts. To accomplish this task it is necessary to need methods to sistematically compute bisimulation functions for classes of transition systems.

Consider two non-linear dynamical systems D^i with $i \in \{1, 2\}$

$$D^i = \begin{cases} \dot{\mathbf{y}}^i & = \mathbf{f}^i(\mathbf{y}^i, \mathbf{I}^i) \\ \dot{\mathbf{x}}^i & = \mathbf{h}^i(\mathbf{y}^i) \end{cases}$$

where $\mathbf{y} \in \mathbb{R}^{n_i}$, $\mathbf{y}^i(0) \in Y_i$ compact subset of $\mathbb{R}^{n_i}$, $\mathbf{I}^i \in U^i$ compact set of $\mathbb{R}^{m_i}$. $\mathbf{x}^i \in \mathbb{R}^p$ assuming that D^1 and D^2 have the same observation space $\mathbb{R}^{p_1} = \mathbb{R}^{p_2} = \mathbb{R}^p$ equipped with the euclidean distance.

From these dynamical systems we define two transition systems $T_i = (Q_i, \Sigma_i, \gamma_i, Q_i^0, O_i, \mathbf{h}^i)$ with

- $Q_i = \mathbb{R}^{n_i}$
- $\Sigma_i = \mathbb{R}^+$
- the transition $\gamma_i(\mathbf{y}^i, t) = \mathbf{y}'^i$ stands iff $\forall s \in [0, t]$, $\mathbf{y}^i(0) = \mathbf{y}^i$, and $\mathbf{y}^i(t) = \mathbf{y}'^i$ such that $\dot{\mathbf{y}}^i(s) = \mathbf{f}^i(\mathbf{y}^i(s), \mathbf{I}(s))$

- $Q_i^0 = Y^i$
- the set of Observations $O^i = \mathbb{R}^p$
- the observation map $\mathbf{h}^i$

Denoting

- $\mathbf{y} = \begin{bmatrix} \mathbf{y}^1 \\ \mathbf{y}^2 \end{bmatrix}$
- $\mathbf{f}(\mathbf{y}, \mathbf{I}^1, \mathbf{I}^2) = \begin{bmatrix} \mathbf{f}^1(\mathbf{y}^1, \mathbf{I}^1) \\ \mathbf{f}^2(\mathbf{y}^2, \mathbf{I}^2) \end{bmatrix}$
- $\mathbf{h}(\mathbf{x}) = \mathbf{h}^1(\mathbf{x}^1) - \mathbf{h}^2(\mathbf{x}^2)$

we can express the following important Theorem (Girard and Pappas, 2005):

Theorem 5.3.13. *Let $p : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}^+$ be a differentiable function with ∇p its gradient. If for all $\mathbf{y} \in \mathbb{R}^{n_1+n_2}$ $p(\mathbf{x})$ satisfies*

$$p(\mathbf{x}) \geq \mathbf{h}(\mathbf{x})^T \mathbf{h}(\mathbf{x}) \tag{5.9}$$

$$\max_{\mathbf{I}^1 \in U^1} \min_{\mathbf{I}^2 \in U^2} \nabla p(\mathbf{x})^T \mathbf{f}(\mathbf{y}, \mathbf{I}^1, \mathbf{I}^2) \leq 0 \tag{5.10}$$

$$\max_{\mathbf{I}^2 \in U^2} \min_{\mathbf{I}^1 \in U^1} \nabla p(\mathbf{x})^T \mathbf{f}(\mathbf{y}, \mathbf{I}^1, \mathbf{I}^2) \leq 0 \tag{5.11}$$

then $V_B = \sqrt{p(\mathbf{x})}$ is a bisimulation function.

If we restrict to the class of autonomous dynamical systems with fixed inputs, so that $\mathbf{f}(\mathbf{y}, \mathbf{I}^1, \mathbf{I}^2) = \mathbf{f}(\mathbf{y})$, it happens that Conditions 5.10 and 5.11 collapse to one.

Finding a good candidate function with the condition of being greater than zero is a difficult task. However, the imposing of the sum of squares condition makes the problem simpler[1], even though, of course, restricting at the same time the possible solutions. In fact the a sum of squares condition implies the positive condition, but the converse is not true.

A multivariate polynomial $p(\mathbf{x})$ is a sum of squares if

$$p(\mathbf{x}) = \sum_{i=1}^{i=S} q_i^2(\mathbf{x})$$

where $q_1(\mathbf{x}), \ldots, q_M(\mathbf{x})$ are polynomials.

The following theorem gives an even simpler formulation of Theorem 5.3.13 if we assume that the vector fields $\mathbf{f}^1(\mathbf{x})$ and $\mathbf{f}^2(\mathbf{x})$ are expressed by polynomials. So the task becomes manageable and can be computed in *semidefinite programming*[2].

Theorem 5.3.14. *(Girard and Pappas, 2005) It is possible to search a bisimulation of the form*

$$V_B(\mathbf{x}) = p(\mathbf{x})$$

assuming the hypotheses of autonomous vector fields $\mathbf{f}^1$ *and* $\mathbf{f}^2$*, and that the observation maps* $\mathbf{h}^1$ *and* $\mathbf{h}^2$ *are vectors of polynomials, the Proposition 5.3.13 reduces to:*

$$p(\mathbf{x}) - \mathbf{h}(\mathbf{x})^T \mathbf{h}(\mathbf{x}) \quad \textit{is a sum of squares} \tag{5.12}$$

$$-\nabla p(\mathbf{x})^T \mathbf{f}(\mathbf{x}) \quad \textit{is a sum of squares} \tag{5.13}$$

[1] It has been shown (see Parrilo, 2003) that the condition "$p(\mathbf{x})$ *is a sum of squares*" is computationally more tractable than $p(\mathbf{x}) \geq 0$.

[2] In particular to accomplish the algorithm subsumes by Theorem 5.3.14 SOSTOOLS Matlab toolbox (Prajna et al., 2002) will be deployed.

The theorem reveals the *Bisimulation Function* algorithm used to find a bisimulation functions: given an expression of the polynomial fixing the terms $a_i(\mathbf{x})$

$$p(\mathbf{x}) = c_1 a_1(\mathbf{x}) + c_2 a_2(\mathbf{x}) + \cdots + c_m a_m(\mathbf{x})$$

If we find coefficients c_i satisfying Theorem 5.3.14, we are able to write a bisimulation function

$$V_B(\mathbf{x}) = \sqrt{p(\mathbf{x})}$$

5.4 Application of the method

In this section we show how to apply these techniques on a toy example before they will systematically be applied in the experiments of Chapter 6. We consider the two dynamical systems

$$D_1 = \begin{cases} \dot{y}_1 &= k \cdot z - y_1 \\ \dot{x} &= -x \\ z_1 &= y_1 \end{cases}$$

and

$$D_2 = \begin{cases} \dot{y}_2 &= -y_2 \\ z_2 &= y_2 \end{cases}$$

The two systems are easily analyzed. D_2 is an equation with only one global stable fixed point in 0. Each solution starting on different initial conditions y_2^0 will eventually go to 0. Similarly the other system eventually approaches the global stable fixed point $(0, 0)$. Fig. 5.5 shows the vector field of D_1. The identity function was chosen as transfer function for the observable variables with the aim of comparing the variables $z_1 = y_1$

against $z_2 = y_2$.

We want to study this toy example with the techniques we explained in thr previous section.

First of all notice that h_1 and h_2 are the identities functions so that the transition system associated to D_1 is a transition system with clean observables, and to one associated to D_2 is a completely observable transition system. This will always be the case of Chapter 6 when comparing original network behaviour against the DMAN interpreter with the corresponding programming codes.

Now, as we need to systematically find polynomial bisimulation functions, independently of the system in exam, we have to choose a general form of multivariate polynomial of the $M-th$ order considering all the possible combinations of the variables. Thus rewriting these combinations as $\{Comb_k^m(\mathbf{q}_1, \mathbf{q}_2)\}_{k=1}^{K_m}$ where[3] $K_m = \begin{pmatrix} m+n_1+n_2-1 \\ m \end{pmatrix} = \frac{(m+n_1+n_2-1)!}{(m)!(n_1+n_2-1)!}$ the multivariate polynomial assumes the form

$$V_M(\mathbf{q}_1, \mathbf{q}_2) = \sum_{m=0}^{M} \sum_{k=1}^{K_m} c_m^k Comb_k^m(\mathbf{q}_1, \mathbf{q}_2) \tag{5.14}$$

Specifically for D_1 and D_2 it assumes for $M = 0$ the form

$$V_0(y_1, x, y_2) = \sum_{k=1}^{K_1} c_0^k Comb_k^0(y_1, z, y_2) = c_0^1$$

[3]It is easy to see that the number combinations of terms of a polynomial of the same degree is equal to a solution of the positive integer solutions of the equation

$$x_1 + x_2 + \cdots + x_N = m$$

that is given by the binomial coefficient $\begin{pmatrix} m+N-1 \\ m \end{pmatrix}$

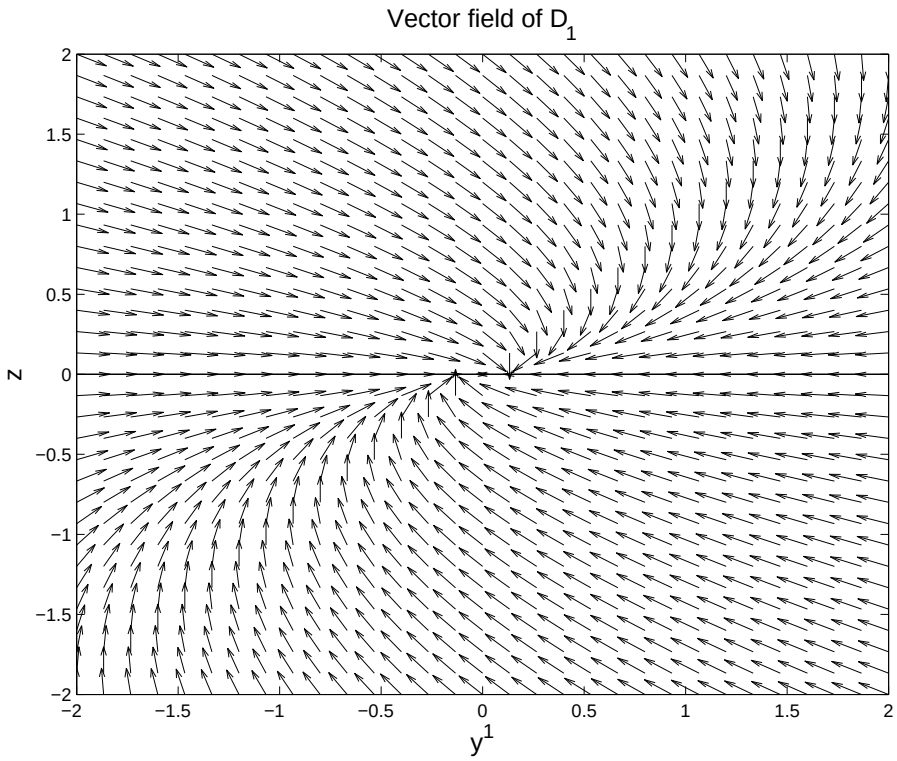

Figure 5.5: D_1 dynamical system vector field

then for $M = 1$

$$V_1(y_1, x, y_2) = \sum_{m=0}^{1} \sum_{k_m=1}^{K_m} c_m^k Comb_k^1(y_1, z, y_2) = c_0^1 + c_1^1 y_1 + c_1^2 y_2 + c_1^3 x$$

and $M = 2$

$$V_2(y_1, x, y_2) = c_0^1 + c_1^1 y_1 + c_1^2 y_2 + c_1^3 x + c_2^1 (y_1)^2 + c_2^2 (y_2)^2 + c_2^3 x^2 + c_2^4 y_1 y_2 + c_2^5 y_1 x + c_2^6 y_2 x$$

It is clear that polynomial becomes very complex augmenting the number of the variables. For our experiments we always chose V_2 polynomials, which let us find good bounds for our bisimulation functions. Also in this sample experiments, we found different polynomials varying the variable k. It is clear from the equation that the higer value k assumes the more distant D_1 and D_2 trajectories are distance. On the other hand for $k = 0$ the systems are bisimilar. Coherently we find the coefficients of V_2 which satisfy the conditions of Proposition 5.3.14 (see Table 5.2). The values which assume V_2 give us a theoretical bound for the distance of the trajectories of the two systems. Moreover the maximum of δ gives us a uniform bound for all the trajectories of D_1 and D_2 (see Table 5.1). This means that if the bound is sufficient for the tasks in consideration, the two systems can be acceptably equivalent. Fig. 5.6 shows a sample comparison of observable trajectories for D_1 and D_2 for $k = 10$, with initial conditions $y_1^0 = y_2^0 = 1$ and $x^0 = 1$. As we can see the bound is inside the theoretical bound in Table 5.1. At the same time we executed pointwise measures of the euclidean distances between trajectories of D_1 and D_2. This measure, indicated with δ_t, expresses at the time

k	$\delta_{\max}$
10	7.4822
9	6.8611
8	6.3338
7	5.5777
6	4.8671
5	4.1851
4	3.6782
3	2.8938
2	2.5492
1	2.0142

Table 5.1: Values of the maximum δ relative to the V_2 bisimulation multivariate polynomial founds, computed for $y_1, y_2, x \in [-1, 1]$ imposing initial condition $y_1^0 = y_2^0$ for variable which we want to compare. Coherently as we expected $\delta_{\max}$ decreases with the values of k.

t how far the points of the observable spaces of D_1 and D_2 are. In Fig. 5.7 δ_t as a function of the time is depicted. It is possible to notice how for the system D_1 and D_2, after an initial increase in distance due the initial perturbation of k, it tends to decrease going to zero when approaching the stable fixed point. In general this is the same behaviour we found in measures of the DMANs interpreting behaviour as it is possible to see in the next Chapter. Thus, in most cases, even in the presence of higher $\delta_{\max}$, the decreasing of the measure δ_t for t approaching to infinity, let the system play still good performance in attractor computation. In the next Chapter combination of measures $\delta_{\max}$ and δ_t are extensively used in the experiments bringing information on the interpreting capabilities of actual DMANs implementations.

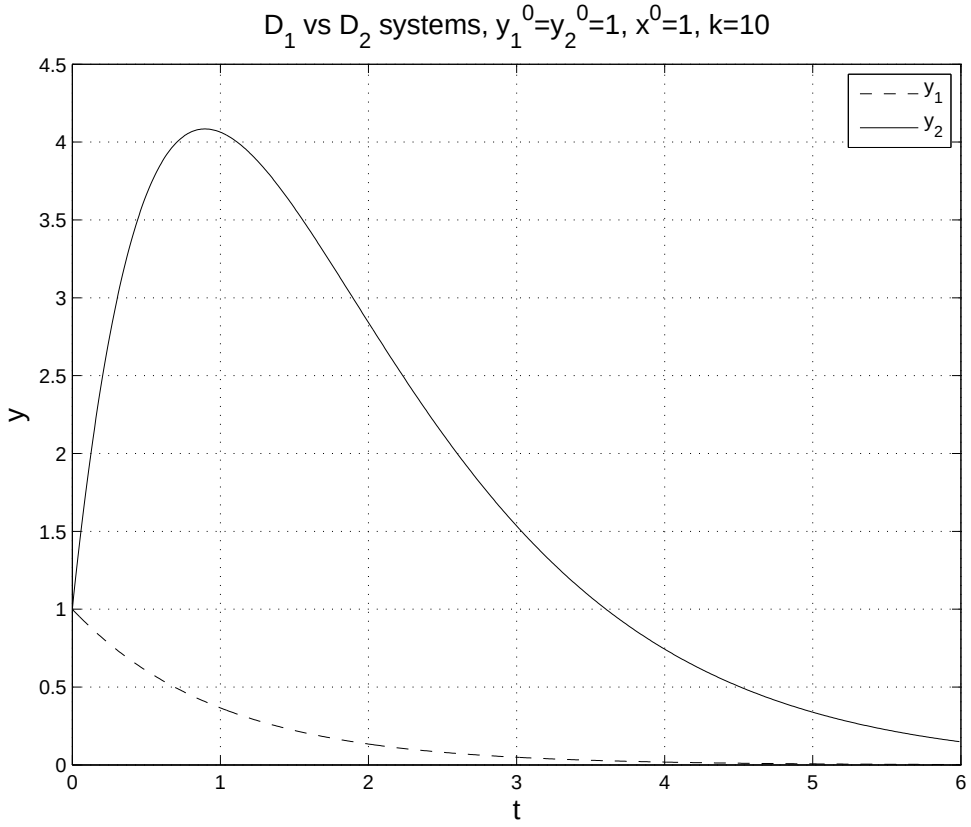

Figure 5.6: Comparing the trajectories of $D_1$and D_2 on the observables y_1 and y_2. The parameter $k = 10$ and the initial conditions are $y_1^0 = y_2^0 = 1$ and $x^0 = 1$.

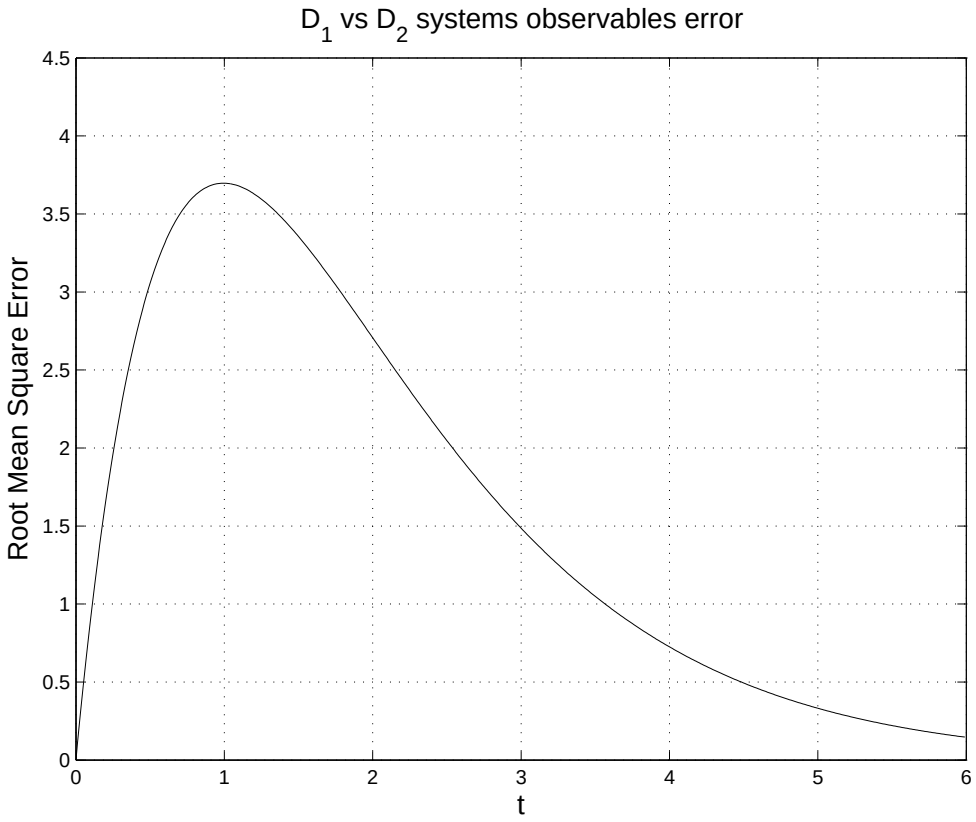

Figure 5.7: δ_t error between the observables of $D_1$and D_2 in the case of parameter $k = 10$ and the initial conditions are $y_1^0 = y_2^0 = 1$ and $x^0 = 1$. When t approaches infinity δ_t approaches zero showing how the observables of D_1 and D_2 computes the same function in terms of attractor computation.

k	c_0^1	c_1^1	c_1^2	c_1^3	c_2^1	c_2^2	c_2^3	c_2^4	c_2^5	c_2^6
10	$0.14413 \cdot 10^{-4}$	$-0.57622 \cdot 10^{-5}$	$0.35332 \cdot 10^{-5}$	$-0.26582 \cdot 10^{-4}$	1	8.7456	32.746	-2	$0.67654 \cdot 10^{-5}$	-15.491
9	$0.12563 \cdot 10^{-4}$	$-0.55813 \cdot 10^{-5}$	$0.3802 \cdot 10^{-5}$	$-0.23336 \cdot 10^{-4}$	1	7.7061	26.956	-2	$0.66117 \cdot 10^{-5}$	-13.412
8	$0.11959 \cdot 10^{-4}$	$-0.59776 \cdot 10^{-5}$	$0.53081 \cdot 10^{-5}$	$-0.23241 \cdot 10^{-4}$	1	7.0293	22.029	-2	$0.71998 \cdot 10^{-5}$	-12.059
7	$0.94014 \cdot 10^{-5}$	$-0.53715 \cdot 10^{-5}$	$0.38464 \cdot 10^{-5}$	$-0.17275 \cdot 10^{-4}$	1	5.7153	16.965	-2	$0.55346 \cdot 10^{-5}$	-9.4307
6	$0.74218 \cdot 10^{-5}$	$-0.49477 \cdot 10^{-5}$	$0.37436 \cdot 10^{-5}$	$-0.13639 \cdot 10^{-4}$	1	4.6723	12.672	-2	$0.4822 \cdot 10^{-5}$	-7.3446
5	$0.57541 \cdot 10^{-5}$	$-0.46037 \cdot 10^{-5}$	$0.36935 \cdot 10^{-5}$	$-0.10599 \cdot 10^{-4}$	1	3.8162	9.0662	-2	$0.43041 \cdot 10^{-5}$	-5.6324
4	$0.46506 \cdot 10^{-5}$	$-0.46512 \cdot 10^{-5}$	$0.40666 \cdot 10^{-5}$	$-0.87178 \cdot 10^{-5}$	1	3.3824	6.3824	-2	$0.40637 \cdot 10^{-5}$	-4.7647
3	$0.30405 \cdot 10^{-5}$	$-0.40549 \cdot 10^{-5}$	$0.38205 \cdot 10^{-5}$	$-0.58479 \cdot 10^{-5}$	1	2.531	3.781	-2	$0.36616 \cdot 10^{-5}$	-3.062
2	$0.19501 \cdot 10^{-5}$	$-0.39019 \cdot 10^{-5}$	$0.39824 \cdot 10^{-5}$	$-0.39824 \cdot 10^{-5}$	1	2.3746	2.3746	-2	$0.3556 \cdot 10^{-5}$	-2.7491
1	$0.15855 \cdot 10^{-5}$	$-0.63453 \cdot 10^{-5}$	$0.63313 \cdot 10^{-5}$	$-0.31586 \cdot 10^{-5}$	1	1.9518	1.2018	-2	$0.59228 \cdot 10^{-5}$	-1.9036

Table 5.2: Coefficients for V_2 bisimulation multivariate polynomial found when varying the perturbation parameter $k \in \{1, \ldots, 10\}$.

6

Experiments and Results: validation of the model

In this Chapter we will show significant experimental evidences exhibiting:

- the possibility of actually obtaining CTRNNs, mul^*, which approximate the ideal structure mul we introduced in Section 4.1;
- the plausibility that a single fixed-weight DMAN G_{mul*} can be *programmed* with auxiliary inputs p_h in order to reproduce the dynamical behaviours of networks G with weight values given by (4.1).
- the robustness of the DMAN obtained under variations of the time scales on which the mul^* network acts in order to evaluate how such changes affect the interpreting capability of our architecture.

The experiments were performed on small CTRNNs which were numerically integrated by means of the forward Euler method exposed in Section 2.3. The integration step size used is $\Delta T = 0.2$ and the time constants of the neurons assume values $\tau_i \geq 1$.

6.1 Ideal mul approximations

In Chapter 4 we explain the ideal behaviour of mul (see Fig. 6.1), for which a $w-$substitution would preserve intact the interpreting capabilities of DMANs. In this section we present the approximation of the ideal mul that we are going to deploy in the experiments.

The first approximation is for the dynamical model of mul shown in 6.1.1, which provides a time-delayed version of mul behaviour (the stability surface of the dynamical $mul^\star$ is the same as the ideal mul). The approximation is needed for the application of Theorem 5.3.14 which implies a polynomial version of a CTRNN system.

The second approximation is about finding an actual CTRNN $mul^\star$, that is a CTRNN which shows the behaviour of the ideal mul. This is achieved by a machine learning algorithm which gives an approximation version of the ideal mul both in time delay and in the values of the fixed point stability surface.

6.1.1 A dynamical $mul^\star$ equation

The first dynamical approximation is provided searching for equations the surface stability of which should be exactly the multiplication of its input, and its approach to fixed point should be a non zero time delay. Such a behaviour is given by the following equation:

$$\tau_{mul} \cdot \frac{dx_{mul}}{dt} = -x_{mul} + a \cdot b \tag{6.1}$$

In fact it is possible to find the solutions of the equation separating the variables

$$\frac{dx_{mul}}{x_{mul} - a \cdot b} = -\frac{1}{\tau_{mul}} \cdot dt$$

and then writing

$$\ln\left(\frac{x_{mul} - a \cdot b}{x^0_{mul} - a \cdot b}\right) = -\frac{1}{\tau_{mul}} \cdot t$$

Thus it is possible to find

$$\frac{x_{mul} - a \cdot b}{x^0_{mul} - a \cdot b} = e^{-\frac{1}{\tau_{mul}} \cdot t}$$

obtaining

$$x_{mul} = a \cdot b + (x^0_{mul} - a \cdot b) \cdot e^{-\frac{1}{\tau_{mul}} \cdot t}$$

and the finally solutions of the flow of 6.1

$$\gamma(x^0_{mul}, t) = a \cdot b + (x^0_{mul} - a \cdot b) \cdot e^{-\frac{1}{\tau_{mul}} \cdot t}$$

The Jacobian is reduced to -1, so from Theorem 2.6.23 only one asymptotically stable fixed point exist. From the condition of Theorem 2.6.22 we obtain.

$$\bar{x}_{mul} = a \cdot b$$

Thus the Equation 6.1 simulates for every initial condition the stability surface of ideal mul, with an approach to its stability surface regulated by the time constant τ_{mul}. For example the $w-$substitution for $NetOne$ equation of this dynamical model 6.1

$$\begin{aligned} \tau \cdot \frac{dy}{dt} &= -y + x_{mul} + I \\ \tau_{mul} \cdot \frac{dx_{mul}}{dt} &= -x_{mul} + w \cdot \sigma(y) \end{aligned}$$

6.1.2 A polynomial CTRNN approximation

We have explained the possibility of comparing two CTRNN systems by using the similarity measure inside bisimulation techniques explained in Chapters 2 and 4.

However the application of Theorem 5.3.14 gives us an effective procedure only in the presence of polynomial equations of the system. The presence of the sigmoid prevents us from applying the similarity algorithm out of the box. So in this section we use regularized least square (see Bishop, 2006) in order to obtain a polynomial approximation for sigmoid function

$$\sigma(x) \approx \sum_{m=0}^{M} c_i \cdot x^m = Pol_M(x)$$

thus a polynomial version of CTRNN Equation (2.1) is obtained:

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{N} w_{ij} Pol_M(y_i - \theta_i) + I_i^e \qquad i \in \{1, \ldots, N\} \tag{6.2}$$

This procedure, given a suitable polynomial order M, lets us approximate as good as we want the behaviour of the sigmoid function in a given interval. We prepare input-output pairs of sigmoid function $\sigma(x)$ in a fixed interval $[x_{min}, x_{max}]$ in order to

M	1	2	3	4	5	6	7	8
c_0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
c_1	0.0244	0.0244	0.0455	$-1.3603e-21$	0.0656	0.0656	0.0848	0.0848
c_2	0	$-2.9296e-20$	$1.6361e-19$	$-3.7452e-05$	$8.4580e-19$	$3.3302e-19$	$3.1874e-18$	$-4.6122e-18$
c_3	0	0	$-3.7452e-05$	$1.0589e-18$	$-1.3787e-04$	$-1.3787e-04$	$-3.2361e-04$	$-3.2361e-04$
c_4	0	0	0	0.0455	$-1.0334e-21$	$5.2234e-23$	$-1.5470e-20$	$3.4859e-20$
c_5	0	0	0	0	$9.6712e-08$	$9.6712e-08$	$5.3551e-07$	$5.3551e-07$
c_6	0	0	0	0	0	$-1.0049e-25$	$1.6259e-23$	$-8.5790e-23$
c_7	0	0	0	0	0	0	$-2.9154e-10$	$-2.9154e-10$
c_8	0	0	0	0	0	0	0	$6.7205e-26$

Table 6.1: Coefficient of $Pol_M(x)$ found with regularized least squares method approximating a sigmoid $\sigma(x)$ in the interval $x \in [-30, 30]$.

M	1	2	3	4	5	6	7	8
$\bar{E}$	0.1845	0.1845	0.1160	0.1160	0.0831	0.0831	0.0630	0.0630
dev	0.0133	0.0133	0.0077	0.0077	0.0046	0.0046	0.0029	0.0029

Table 6.2: Mean error $\bar{E}$ and standard deviation dev for Pol_M.

apply the regression algorithm. Table 6.1 shows the coefficients found for a Pol_M in an interval $[-30, 30]$. Figure 6.2 shows the behaviour of this approximation for $M = 1$, $M = 3$, $M = 8$ compared to sigmoid function $\sigma(x)$. Table 6.2 shows the mean errors $\bar{E}$ of the different Pol_M $\bar{E}$ decreasing with the order of the polynomial. In the application of the bisimilarity procedure we chose $M = 3$ and the corresponding $Pol_3(x)$. Of course the two systems can be considered comparable as far as the potential $\mathbf{y}$ of the CTRNN does not take values outside $[x_{min}, x_{max}]$. However whenever it happens, it is always possible to regress a new $Pol_M(x)$ which approximate $\sigma(x)$ in a wider range.

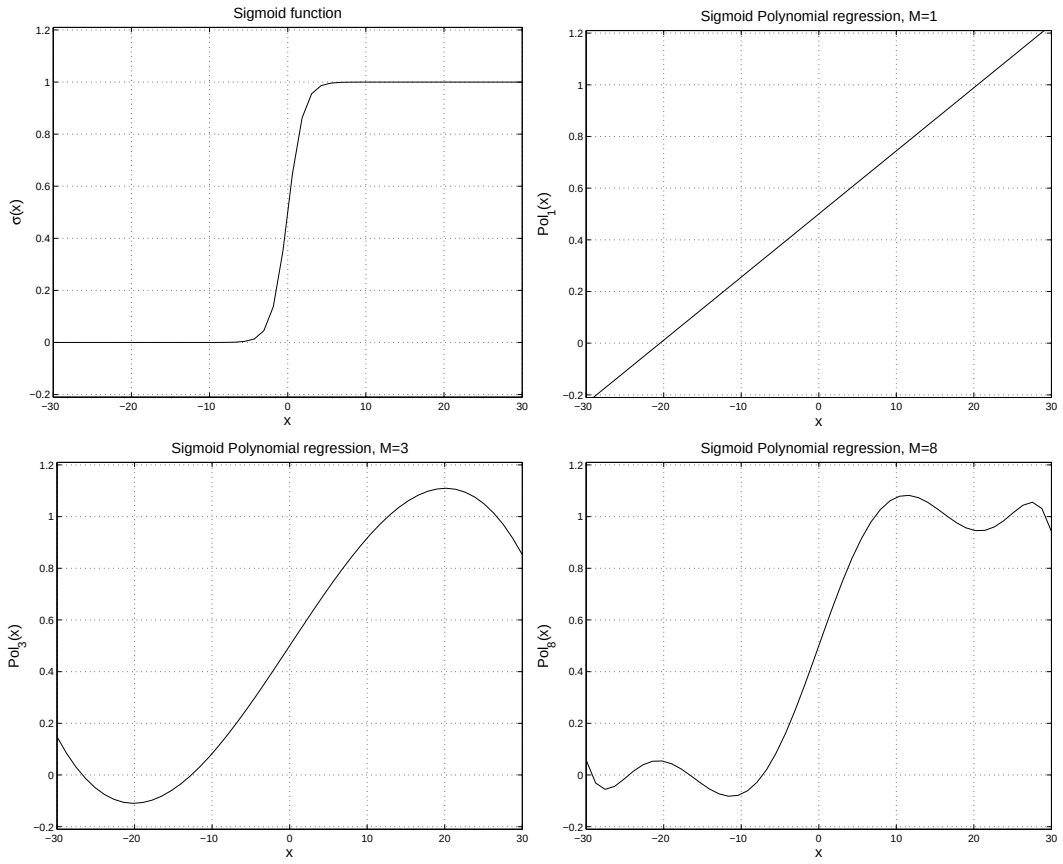

Figure 6.2: Sigmoid Function $\sigma(x)$ versus the polynomial approximations $Pol_1(x)$, $Pol_3(x)$ and $Pol_8(x)$.

6.1.3 The learning algorithm: Differential Evolution

The most common techniques for training both feed-forward and recurrent neural networks are variations of the gradient descent techniques. Various variations of the backpropagation have been investigated in approximating the time evolution of a recurrent neural network as a sequence of static networks using gradient methods realizing a plethora of approaches (see e.g. Lapedes and Farber, 1986; Pineda, 1987; Almeida, 1990; Pearlmutter, 1995; Steil, 2004). However one of the known problems of backpropagation is the possibility of entrapping into local minima during the process of optimization. On the other hand evolutionary techniques are biologically inspired population based machine learning techniques known to be an efficient and effective means of learning, as much as they provide an intrinsically randomness in the search of the solutions. Thus they allow a wider exploration of the space of the solution values.

Consequently in this thesis we decided to apply, in searching for CTRNN mul^*, an evolutionary technique which can be viewed as an evolutionary version of gradient descent, called Differential Evolution (DE) (Price et al., 2005) here briefly described, before being applied to CTRNN cases.

DE is a stochastic, population-based evolutionary algorithm. Fast convergence and ease of use due to few control parameters are distinctive features of this type of algorithm. DE addresses a generic optimization problem with m real parameters by starting with a randomly initialized population consisting of n individuals, each made up of m real values. Subsequently, the population is updated from a generation to the next one by means of many different transformation schemes commonly named as *strategies*. In all of these strategies DE generates new

individuals by adding to an individual a number of weighted difference vectors between couples of population individuals.

The strategy adopted here can be referred to as $DE/best/v/bin$: one perturbs the best individual x_{best} by using v difference vectors and applies binomial crossover. In greater detail, for each i-th individual x_i, x_{best}, corresponding to the best one in the current population, is selected and $2v$ integer numbers $r_1, r_2, \ldots, r_{2v}$ in $[1, n]$, differing from one another and different from i, are randomly generated. Furthermore, another integer number l in the range $[1, m]$ is randomly chosen. Then, starting from the i-th individual a new trial one x'_i is generated, the generic j-th component of which is given by:

$$x'_{i,j} = x_{best,j} + F \cdot [(x_{r_1,j} - x_{r_2,j}) + \cdots + (x_{r_{2v-1},j} - x_{r_{2v},j})] \quad (6.3)$$

provided that either a random real number ρ in $[0.0, 1.0]$ is lower than a value CR (parameter of the algorithm, in the same range as ρ) or the position j at issue is exactly l. If neither condition is verified then a copy process takes place: $x'_{i,j} = x_{i,j}$.
F, a real constant factor in $[0.0, 1.0]$, is a parameter of the algorithm which controls the magnitude of the differential variation $F \cdot [(x_{r_1,j} - x_{r_2,j}) + \cdots + (x_{r_{2v-1},j} - x_{r_{2v},j})]$.

This new trial individual x'_i is compared with the i-th individual in current population and, if it turns out to be fitter, x'_i is substituted in the next population, otherwise the $i-$th individual survives and is copied into the new population. This basic scheme is repeated for a maximum of g_{max} generations.

In order to clarify how $DE/best/v/bin$ works, we present its pseudocode. If s is the size of the population P_g at generation g and the operators $\oplus$, $\ominus$, $\odot$ respectively denote vector addition

Algorithm 6.1 DE/best/v/bin in C-like pseudocode

```
Initialize and evaluate population P_0
for g = 0; g < g_max; g++ do
  for i = 0; i < s; i++ do
    Select the individual x_i
    Select as parents x_best and 2v individuals randomly, all
    different
    {Create an initial candidate:}
    x'_i = x_best ⊕ F ⊙ [⨁_{k=1}^{v} (x_{r_{2k-1}} ⊖ x_{r_{2k}})]
    {Create a final candidate by crossing over the genes of x'_i
    and x_i:}
    Randomly select an integer l ∈ [1, m]
    for j = 0; j < m; j++ do
      Randomly select a real ρ ∈ [0.0, 1.0]
      if ρ > CR and j ≠ l then
        x'_{i,j} = x_{i,j}
      end if
    end for
    Evaluate the candidate x'_i
    if x'_i is fitter then x_i then
      x_i = x'_i
    end if
  end for
  {Substitute the old population with the new one}
  P_{g+1} = P_g
end for
```

and subtraction and scalar multiplication, then DE algorithm pseudocode can be written as in Algorithm 6.1.

Cusp Point Learning

The DE technique was introduced for CTRNN parameter learning in (De Falco et al., 2008; Price et al., 2005). Here we tested the efficacy of CTRNN training by DE on a sample experiment in which the process finds numerical solutions for the

cusp point outlined in Section 5.1.2. In what we named *cusp point learning*, we show the possibility of finding "exact" solutions, without limitation due to encoding resolution (*granularity*), finding very sparse solutions, very difficult to reach with a-priori fixed intervals (*boundedness*). In both cases, we used a $DE/best/2/bin$ strategy to train networks. Parameters ruling DE algorithm (Table 6.3) were assigned experimentally via a training trial.

Experiment	**s**	**F**	**CR**	**initial range**	$\mathbf{g_{max}}$
Cusp point learning	30	0.5	0.8	$(-100, 100)$	3000
Sequence generator task	30	0.7	0.8	$(-100, 100)$	2500

Table 6.3: DE parameter settings

Let us consider a CTRNN made up of a single self-connected neuron. The equation of the system is given by (5.1) where for simplicity we set the time constant $\tau = 1$.

Notice that no elementary expression for the solution of (5.1) exists, but we achieved a complete qualitative description of its dynamics in 5.1.2 describing its limit sets, including their stability and their dependence on the parameters, as well as the bifurcations that can occur as the parameters are varied. For each θ such system has a *cusp point*, that is the only bifurcation point in which the system undergoes a pitchfork bifurcation, as we explained in Section 5.1.2. All other bifurcation points are saddle-node bifurcations (*cusp curve*). The two branches of the cusp intersect in the cusp point and satisfy:

$$\begin{aligned} I &= \bar{y}_1(w,\theta) - w\sigma\left(\bar{y}_1(w,\theta) - \theta\right) \\ I &= \bar{y}_2(w,\theta) - w\sigma\left(\bar{y}_2(w,\theta) - \theta\right) \end{aligned} \tag{6.4}$$

where $\bar{y}_1(w,\theta)$ and $\bar{y}_2(w,\theta)$ are fixed point expressions as a

function of w and θ that satisfy the two conditions, (5.4) and (5.5):

To evaluate each candidate (I', w') we let each parametrized system evolve for a *sufficient* time T so that we can consider the approximation $y'(T) \approx \bar{y}'$. Then we choose a fitness function F_{CP} for the cusp point learning

$$F_{CP}(y'(I', w')) = f_{fixed} + f_{tangent} + f_{cusp}$$

with 3 contributes obtained considering the case $\theta = 0$:

- $f_{fixed} = |-\bar{y}' + w'\sigma(\bar{y}') + I'|$ from condition (5.4);
- $f_{tangent} = |w'\sigma(\bar{y}')(1 - \sigma(\bar{y}')) - 1|$ from condition (5.5);
- $f_{cusp} = |\bar{y}_1(w', 0) - w'\sigma(\bar{y}_1(w', 0)) - \bar{y}_2(w', 0) + w'\sigma(\bar{y}_2(w', 0))|$ from the intersection of the curves in (6.4).

Average and standard deviation values found for (I, w) in 10 runs using the DE algorithm are reported in Table 6.4. These values are absolutely close to the coordinates $(\tilde{I}, \tilde{w}) = (-2, 4)$ of the cusp point which are formally inferred from the condition of the intersection and putting $\theta = 0$. Furthermore, the best and the worst values found for the parameters in runs, respectively,

$$(I_{best}, w_{best}) = (-2.000016, 4.000061)$$

and

$$(I_{worst}, w_{worst}) = (-2.00056, 4.0011)$$

indicate that parameter values computed in every run are very close to each other and this also proves the general efficacy of the approach. The plots in Figure (6.3) show fitness trend as a

Parameter	Average	Standard Deviation
I	-2.00015	$1.6 \cdot 10^{-4}$
w	4.0003	$3.1 \cdot 10^{-4}$

Table 6.4: Average and standard deviation values of parameters I and w computed in 10 runs for the cusp point learning experiment.

function of the generation number for average, best and worst case. It is worth underlining the constant and smooth decrease which suggests a gradual and continuous learning improvement as the generation number grows.

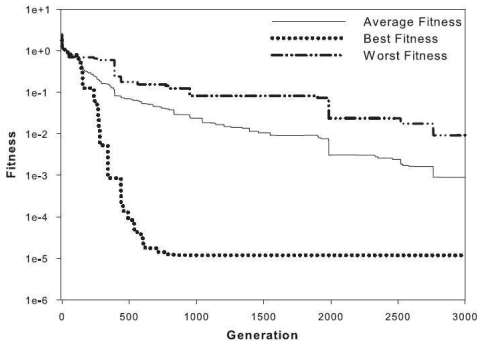

Figure 6.3: Cusp point learning: fitness plots of runs corresponding to the average, worst and best solutions as a function of the generation number.

6.1.4 CTRNN mul^* network

In order to obtain mul^* we used the presented differential evolution learning algorithm. This learning algorithm was made to run on populations of 30 small CTRNNs with $\tau_i = 1$. We

choose as fitness function the distance

$$d = \frac{1}{2K} \sum_{i=1}^{K} \left(\sigma(y^i(T_{eval}) - a^i \cdot b^i\right)^2$$

where $a^i, b^i \in (0,1)$ are K random input values and $\sigma\left(y^i(T_{eval})\right)$ is the corresponding network output value calculated at the evaluation time T_{eval}. The networks we considered are randomly initialized, fed with two data inputs $a^i, b^i \in (0,1)$, and their output is read on the output neuron after a number of integration steps equal to $s = 300$. This means that we evaluate networks after a time $T_{eval} = s \cdot \Delta T$. In such a way we reward networks able to reach the desired stable fixed point $a^i \cdot b^i$ independently of the initial condition of the internal neurons. We made different evolution runs on different sized networks. The procedure is capable of obtaining suitable CTRNN mul^*. The smallest network with a good approximate behaviour (measured by the fitness value) that was found is composed of three neurons. Table 6.5 shows the weights of this CTRNN.

$w_{11} = -2.719$	$w_{21} = -22.93 \cdot 10^4$	$w_{31} = 1.119$
$w_{12} = -4.132$	$w_{22} = 11.49 \cdot 10^4$	$w_{32} = -1.820$
$w_{13} = -11.713$	$w_{23} = 28.68 \cdot 10^4$	$w_{33} = -1.994$
$w_{14} = 8.186$	$w_{24} = -6.711 \cdot 10^4$	$w_{34} = 2.988$
$w_{15} = 1.796$	$w_{25} = 9.887 \cdot 10^4$	$w_{35} = -3.691$

Table 6.5: mul^* network weights. The network is composed of three neurons numbered with $i \in \{1,2,3\}$ fully interconnected with weights w_{ij} with $j \in \{1,2,3\}$. The output neuron is 1. Each neuron i receives two inputs $x_4 = a$ and $x_5 = b$ respectively weighted by w_{i4} and w_{i5}.

In Fig. 6.4 the stability surface closeness of the ideal mul to

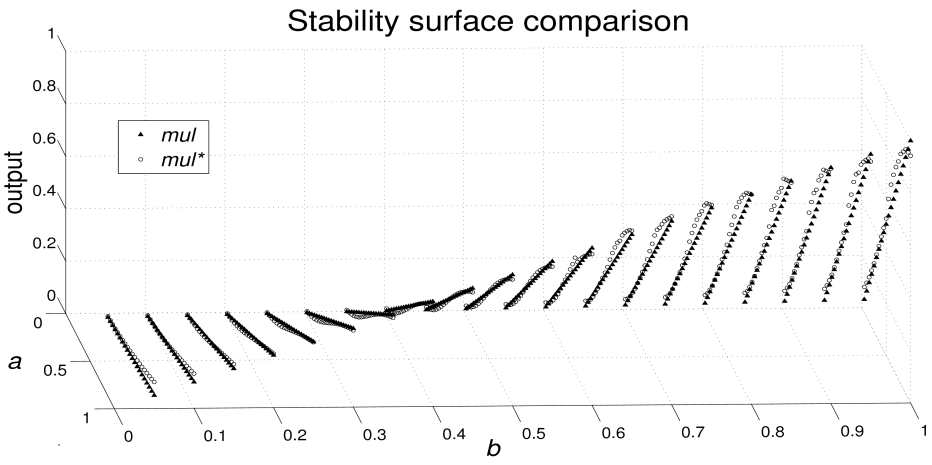

Figure 6.4: Stability surfaces of the output neuron of mul and mul^* as a function of $a, b \in (0, 1)$. Stable equilibrium points of mul are shown as squares, stable equilibrium points of $mul^*$are shown as stars.

the experimental mul^* is shown.

6.2 Single neuron DMAN

As a first case of programmable DMAN we begin our study considering a single neuron with a self-connection with weight $w \in (min = 0, max = 10)$. Let us call this small network $NetOne$; $w-$substituting $NetOne$, of course using the actual mul^*, we construct the DMAN $NetOne_{mul^*}$ which is fed with just one programming input $p_w = (w - min)/(max - min) \in (0, 1)$. In Fig. 6.5 both $NetOne$ and $NetOne_{mul^*}$ are shown. $NetOne$ equation is

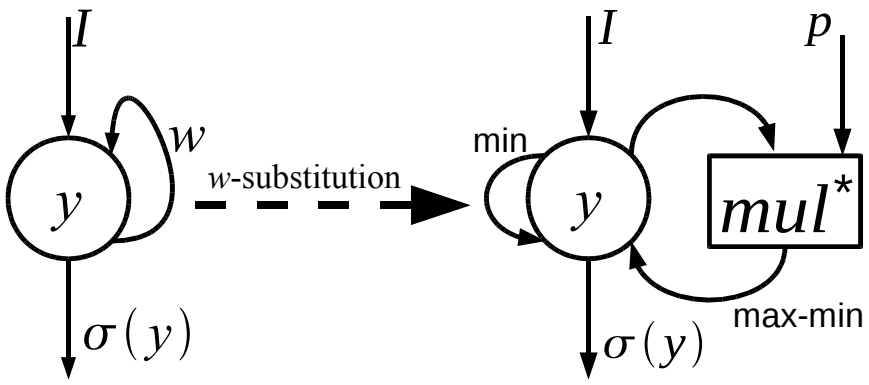

Figure 6.5: The $w-$substitution applied to $NetOne$ (on the left) produces $NetOne_{mul^*}$ (on the right).

$$\tau \frac{dy}{dt} = -y + w\sigma(y) + I \tag{6.5}$$

where the threshold θ is set to 0. We performed a study of the qualitative behaviour of such a small network in Section 5.1.2: the variation of the two parameters w and I modifies the phase portrait of the network. Special values of the parameters exist for which the system undergoes bifurcations. In particular, a qualitative change in behaviour occurs as w passes through value 4. While (6.5) exhibits a global stable equilibrium point when $w < 4$ (see Fig. 6.6 (a)), it exhibits three equilibria for a range of I values when $w > 4$ (see Fig. 6.6 (b)). In the latter case, for I values outside this range, (6.5) exhibits a global stable equilibrium point, while for I belonging to this interval the outer two equilibria are stable and the inner one is unstable.

Here the interesting point is that the fixed-weight network

$NetOne_{mul^*}$ can be programmed in order to obtain the two qualitatively different behaviours by suitably choosing p_w. Figures 6.7, 6.8 and 6.9 show the equilibria of $NetOne$ as a function of I at three different values of w, $w_1 = 3$ (Figure 6.7 (a)), $w_2 = 5$ (Figure 6.8 (a)) and $w_3 = 8$ (Figure 6.9 (a)), compared with the numerically computed equilibria of the fixed-weight $NetOne_{mul^*}$ (restricted to the neuron in $NetOne$) fed with the auxiliary input $p_{w_1} = 0.3$, $p_{w_2} = 0.5$ and $p_{w_2} = 0.8$ (Figure 6.7, 6.8 and 6.9 (b)), respectively .

Thus, the network $NetOne_{mul^*}$ exhibits a global stable equilibrium point when fed with programming input p_{w_1}, while it exhibits three equilibria (one is unstable and two are stable) when the programming inputs are p_{w_2} or p_{w_3}. As a consequence the $NetOne_{mul^*}$ fed with the programming inputs p_{w_1}, $p_{w_2}$and p_{w_3} behaves as virtual $NetOne$ networks with weights w_1, $w_2$and w_3, respectively .

6.3 Programmable *nand* - *or* DMAN

In the previous experiment we showed how the qualitative shape of behaviours can be programmed by means of mul^* on a single neuron. Here we show how a more "quantitative" functionality can be programmed, building a fixed-weight network $Bool_{mul^*}$ which can be programmed to behave, in turn, as the standard binary Boolean functions $nand : \{0,1\} \times \{0,1\} \longrightarrow \{0,1\}$ and $or : \{0,1\} \times \{0,1\} \longrightarrow \{0,1\}$, where we interpret Boolean values as reals.

The first step is to build two networks, $NetAND$ and $NetOR$, to implement the Boolean functions respectively $nand$ and or. This means that if we call $y^{NetNAND}$ and y^{NetOR} the potential

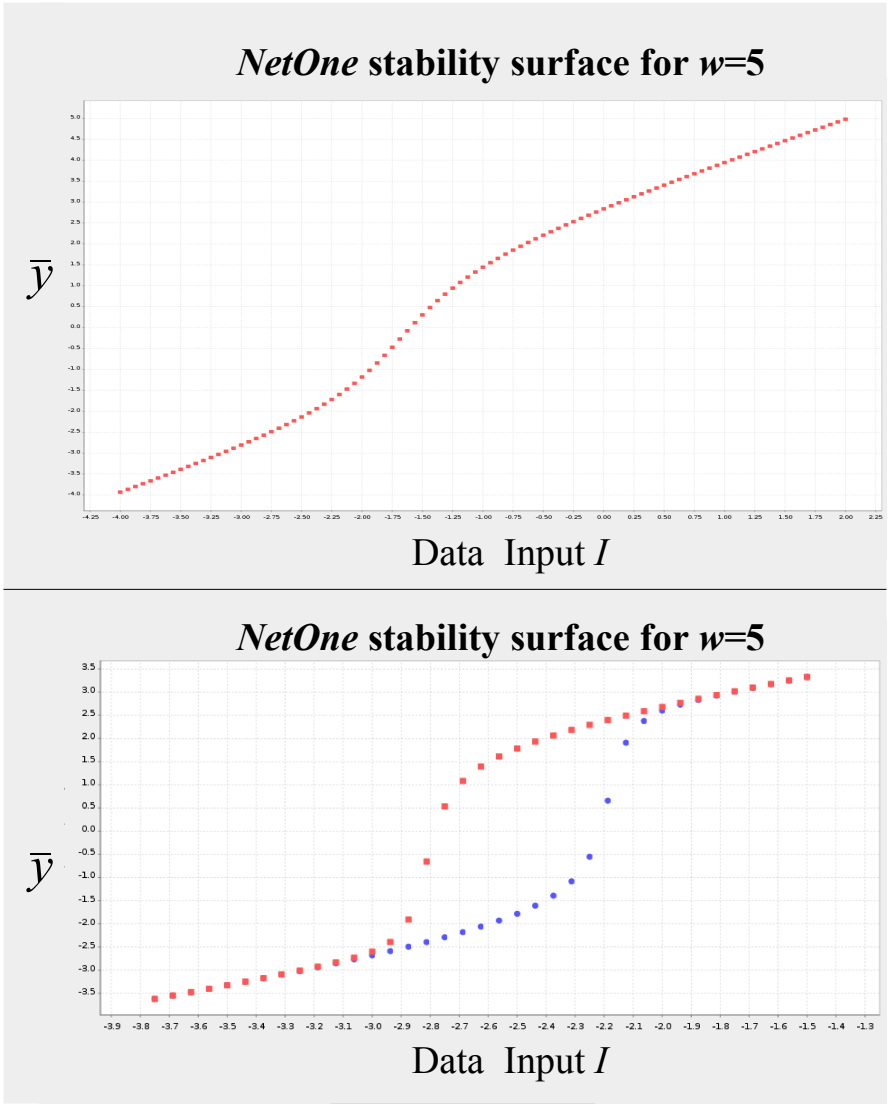

Figure 6.6: In panel (a) of the figure the numerically computed stability surface of *NetOne* for $w = 3$ as a function of I is shown. In this case we have a global stable equilibrium point. The stability surface for $w = 5$ as a function of I is shown in the panel (b). In this case we have two stable equilibrium points inside a range of I values, while for I values outside this range there is a global stable equilibrium point

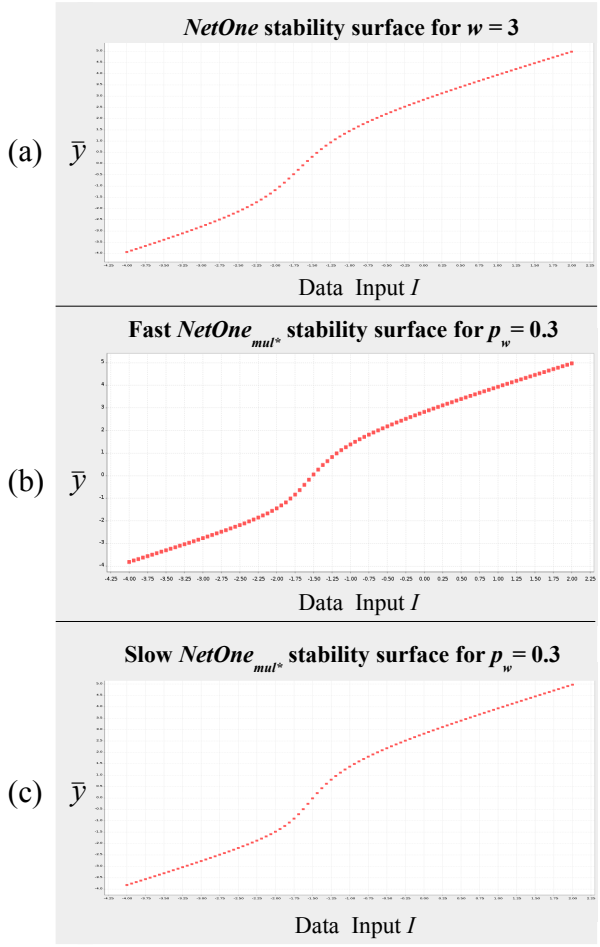

Figure 6.7: The stability surface of $NetOne$ for $w = 3$ as a function of I is shown in panel (a). In the lower part of the figure stability surfaces of $NetOne_{mul^*}$ for the programming input $p_w = 0.3$ as a function of I are shown. In panel (b), the time constants τ_i for the neurons of mul^* have been set one order of magnitude less than that of $NetOne$. In panel (c), the time constants τ_i are two orders of magnitude less than that of $NetOne$. In both cases, we obtain a single equilibrium point. Notice that the shape of the stability surfaces are very similar to that of $NetOne$, indicating that already the first choice of time scale is satisfactory.

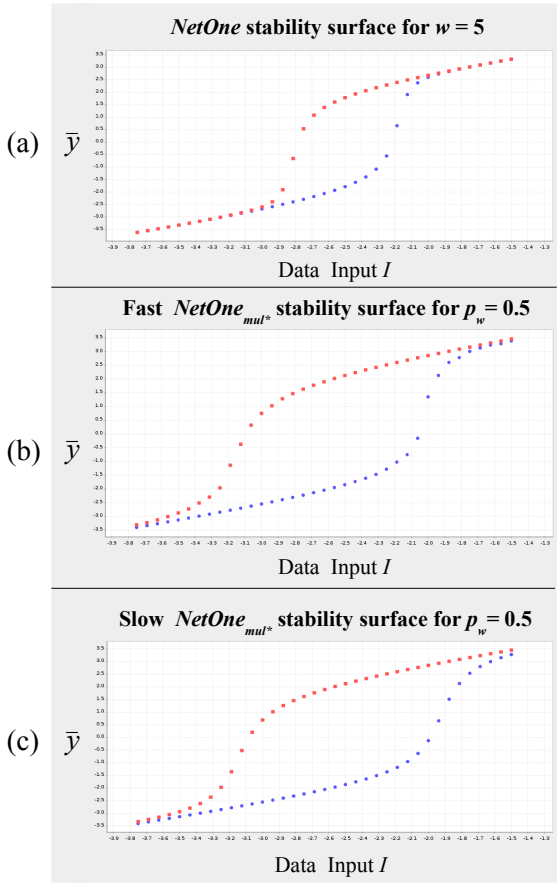

Figure 6.8: The stability surface of $NetOne$ for $w = 5$ as a function of I is shown in panel (a). In the lower part of the figure stability surfaces of $NetOne_{mul^*}$ for the programming input $p_w = 0.5$ as a function of I are shown. In panel (b), the time constants τ_i for neurons of mul^* have been set one order of magnitude less than that of $NetOne$. In panel (c), the time constants τ_i are two orders of magnitude less than that of $NetOne$. In both cases, we obtain two stable equilibrium points inside a range of I values, while for I values outside this range there is a global stable equilibrium point. Notice that the shape of the stability surfaces is very similar to that of $NetOne$, indicating that already the first choice of time scale is satisfactory.

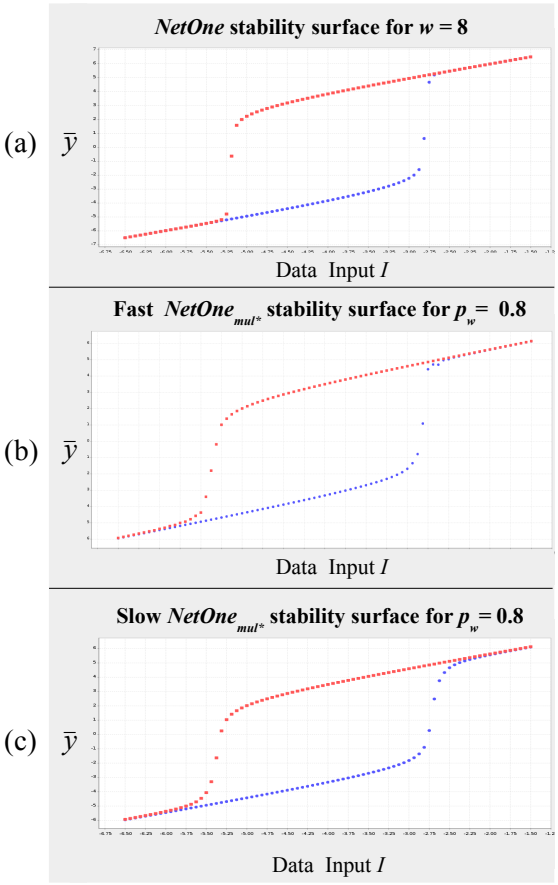

Figure 6.9: The stability surface of $NetOne$ for $w = 8$ as a function of I is shown in panel (a). In the lower part of the figure stability surfaces of $NetOne_{mul^*}$ for the programming input $p_w = 0.8$ as a function of I are shown. In panel (b), the time constants τ_i for neurons of mul^* have been set one order of magnitude less than that of $NetOne$. In panel (c), the time constants τ_i are two orders of magnitude less than that of $NetOne$. In both cases, we obtain two stable equilibrium points inside a range of I values, while for I values outside this range there is a global stable equilibrium point. Notice that the shape of the stability surfaces is very similar to that of $NetOne$, indicating that already the first choice of time scale is satisfactory.

of the output neurons of $NetAND$ and $NetOR$, respectively, and we refer to the inputs of the networks as I_1 and I_2, then their behaviours should be as follows:

$$\begin{aligned} \left|\sigma(\bar{y}^{NetNAND}(I_1, I_2)) - nand(I_1, I_2)\right| &< \delta \\ \left|\sigma(\bar{y}^{NetOR}(I_1, I_2)) - or(I_1, I_2)\right| &< \delta \end{aligned} \tag{6.6}$$

within a threshold $\delta \leq 0.5$.

Our construction directly follows the theory exposed in Chapter 2.

6.3.1 $NetNOT$ cabling

Firstly we build an auxiliary network of only one neuron, $NetNOT$, which simply computes a Boolean function not. Given the equation for one neuron:

$$\dot{y} = -y + w\sigma(y) + I \cdot k$$

the fixed points $\bar{y}$ in this equation are given by the condition:

$$-y + w\sigma(y) + I \cdot k = 0$$

Building the NOT functions means to find parameters $\tilde{w}$ and $\tilde{k}$ so that for two inputs $I_1$and I_2 there exist $\bar{y}^1$ and $\bar{y}^2$

$$I_1 \cdot \tilde{k} - \bar{y}^1 + \tilde{w}\sigma\left(\bar{y}^1\right) = 0 \implies \sigma\left(\bar{y}^1\right) \approx 1$$

$$I_2 \cdot \tilde{k} - \bar{y}_2 + \tilde{w}\sigma\left(\bar{y}^2\right) = 0 \implies \sigma\left(\bar{y}^2\right) \approx 0$$

As we want to create a Boolean function it should happen that $I_1 = 0$ and $I_2 = 1$. Substituting we obtain

$$\bar{y}_1 = \tilde{w}\sigma\left(\bar{y}_1\right)$$

$$\bar{y}_2 - \tilde{k} = \tilde{w}\sigma\left(\bar{y}_2\right)$$

It is worth noting that the output of the σ function could not be exactly 1 or 0 as we expect by a Boolean function; however we can find two fixed points next to them with the desired degree of approximation. The condition $\sigma\left(\bar{y}_1\right) \approx 1$ means that we want to find and ϵ_1 such that $|\sigma(\bar{y}_1) - 1| \leq \epsilon_1$. For example if $\epsilon_1 = 10^{-3}$ it is sufficient to have a fixed point $\bar{y}_1 \geq 7 \implies \sigma(\bar{y}_1) \geq 0.9990889$, hence $|1 - \sigma(\bar{y}_1)| \leq 0.0009111 < \epsilon_1$. Moreover as $\bar{y}_1 \approx \tilde{w}$ consequently the parameter to choose should be $\tilde{w} > 7$. From the condition $\sigma\left(\bar{y}_2\right) \approx 0$ we should find an ϵ_2 such that $|\sigma(\bar{y}_2) - 0| \leq \epsilon_2$, for example $\bar{y}_2 \leq -7 \implies \sigma\left(\bar{y}_2\right) \leq 0.0009111$, hence $|\sigma(\bar{y}_2) - 0| \leq 0.0009111 < \epsilon_2$. But the second condition implies $\bar{y}_2 \approx \tilde{k}$ so the parameter to choose is $\tilde{k} < -7$. For symmetry ($\epsilon_1 = \epsilon_2$) it is possible to set $\tilde{w} = -\tilde{k} = b$ obtaining the $NetNOT$ cabling in Fig. 6.10.

6.3.2 $NetNAND$ cabling

We want to find a network of two neurons, $NetNAND$ capable of showing the properties of the Boolean function *nand* when external input I_1 and I_2 are given to it. We will show how it can be done with two neurons. Firstly we take a $NetNOT$ as we explained in subsection 6.3.1:

$$\dot{y}_2 = -y_2 + c_2\sigma(y_2) - c_2 I_2$$

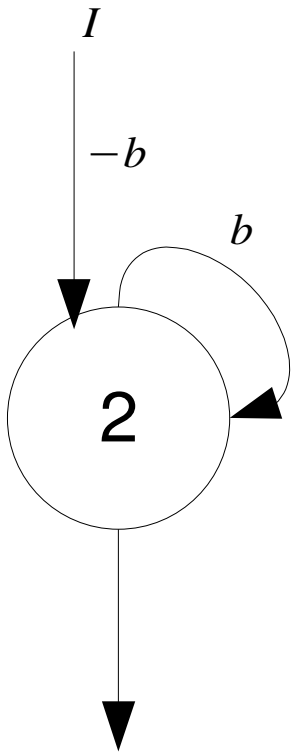

Figure 6.10: *NetNOT* cabling. The choice of b allows the approximation to be as good as necessary.

Which causes the output of this neuron to invert the value of the input I_2. Thus supposing we choose c_2 good enough as we can approximate $\sigma(\bar{y}_2) \approx 0$ when $I_2 = 1$ and $\sigma(\bar{y}_2) \approx 1$ when $I_2 = 0$.

Now we can add an output neuron from which the resulting $NetNAND$ fixed point computation will be read. This neuron takes a connection from the $NetNOT$ neuron above and gives the result as a stable fixed point:

$$\dot{y}_1 = -y_1 + w_{12}\sigma(y_2) + k_1 I_1 + I_3$$

In fact the fixed points of this equation are given by

$$\bar{y}_1 = w_{12}\sigma(\bar{y}_2) + k_1 I_1 + I_3$$

and we can study the four cases

1. $I_1 = 0$, $I_2 = 0$ implies $\bar{y}_1^{00} \approx w_{12} + I_3$
2. $I_1 = 0$, $I_2 = 1$ implies $\bar{y}_1^{01} \approx I_3$
3. $I_1 = 1$, $I_2 = 0$ implies $\bar{y}_1^{10} \approx w_{12} + k_1 + I_3$
4. $I_1 = 1$, $I_2 = 1$ implies $\bar{y}_1^{11} \approx k_1 + I_3$

We impose I_3 as a fictitious static input which allows to have a positive value of $\bar{y}_1^{01}$ good enough in order to have $\sigma(\bar{y}_1^{01}) \approx 1$. Suppose we choose $I_3 = c_1$. Then to satisfy the fourth condition we can choose $k_1 = -2c_1$ in order to get $\bar{y}_1^{11} \approx -c_1$ and a good approximation of $\sigma(\bar{y}_1^{11}) \approx 0$. Then to satisfy the third condition we choose to get $w_{12} = 2c_1$ in order to have $\bar{y}_1^{10} \approx c_1$ so as to have $\sigma(\bar{y}^{10}) \approx 1$. The last condition $\sigma(\bar{y}^{00}) \approx 1$ result satisfied because we have $\bar{y}_1^{00} = 3c_1$.

In summary the equations of $NetNAND$ are given by (see Fig. 6.12):

$$\dot{y}_1 = -y_1 + 2c_1 \cdot \sigma(y_2) - 2c_1 I_1 + c_1$$

$$\dot{y}_2 = -y_2 + c_2 \cdot \sigma(y_2) - c_2 \cdot I$$

As good as we choose the parameters c_1 and c_2, a better approximation of the output 0 and 1 of the Boolean function *nand* we have.

6.3.3 $NetOR$ cabling

The same reasoning spent for $NetNAND$ can be applied in order to construct a network of two neurons which approximately compute as a fixed point computation a Boolean *or* function. This is again achieved by letting one neuron realizing the $NetNOT$ equation

$$\dot{y}_2 = -y_2 + c_2\sigma(y_2) - c_2 I_2$$

and then identically by cabling the second neuron in same manner as the $NetNAND$ case. In this case we achieved the right parameter values by satisfying the four conditions given in the previous subsection in order to have an *or* function.

Again we set I_3 as fictitious static input which allows to have a positive value of $\bar{y}_1^{01}$ good enough in order to have $\sigma(\bar{y}_1^{01}) \approx 1$. If we choose $I_3 = c_1$, then to satisfy the first condition we can choose $w_{12} = -2c_1$ in order to get $\bar{y}_1^{00} \approx -c_1$, resulting in a good approximation of $\sigma(\bar{y}_1^{11}) \approx 0$. Then to satisfy the third condition we choose to get $k_1 = 2c_1$ in order to have $\bar{y}_1^{10} \approx c_1$ so that $\sigma(\bar{y}_1^{10}) = 1$. The last condition results satisfied with $\bar{y}_1^{11} = 3c_1$.

In summary the equations of $NetOR$ are given by (see Fig. 6.11):

$$\dot{y}_1 = -y_1 - 2c_1 \cdot \sigma(y_2) + 2c_1 I_1 + c_1$$

$$\dot{y}_2 = -y_2 + c_2 \cdot \sigma(y_2) - c_2 \cdot I$$

As good as we choose the parameters c_1 and c_2, a better approximation of the output 0 and 1 of the Boolean function *nand* we have.

6.3.4 $Bool_{mul^*}$DMAN construction

We constructed $NetNAND$ and $NetOR$ such that suitable choices of the parameters c_1 and c_2 realize good approximation of the Boolean functions *nand* and *or*, respectively. This is achieved satisfying the Equation (6.6); in particular for our experiments we set $2c_1 = c_2 = 5$, allowing $\delta = 0.1$.

We built $Bool_{mul^*}$ by applying the w-substitution on the only two connections which differ in the two networks as shown in Fig. 6.13. Then as shown in Table 6.6 when we set the programming inputs as $\mathbf{p}^{NetNAND} = [p_1^{NetNAND} = 0, p_2^{NetNAND} = 1]$ and $\mathbf{p}^{NetOR} = [p_1^{NetOR} = 1, p_2^{NetOR} = 0]$, which according to (6.6), with $min = -5$ and $max = -10$, codify the required weights $-2c_1$ and $2c_1$ respectively, $Bool_{mul^*}$ is able to reproduce the behaviour of the networks $NetAND$ and $NetOR$. Notice that $Bool_{mul^*}$ simulates $NetNAND$ and $NetOR$ within $\delta = 0.3$.

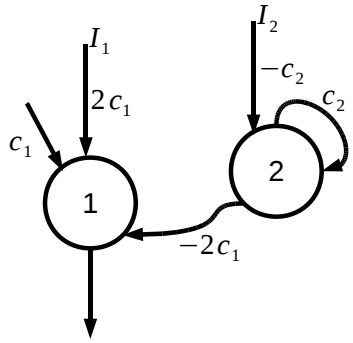

Figure 6.11: The cabling of $NetOR$. Suitable choices of parameters c_1 and c_2 allow steady states closer to the desired stable equilibrium points 0 and 1.

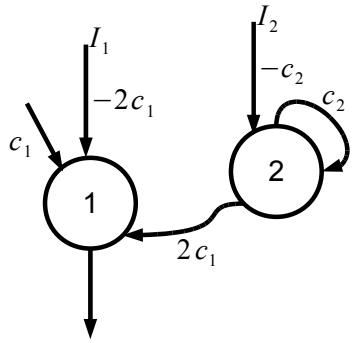

Figure 6.12: The cabling of $NetNAND$. Suitable choices of parameters c_1 and c_2 allow steady states closer to the desired stable equilibrium points 0 and 1.

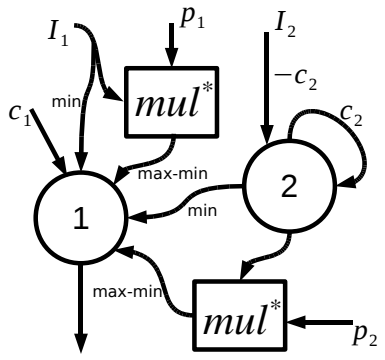

Figure 6.13: The cabling of $Bool_{mul^*}$ network, obtained by the pulling out of two connections with respect to $NetNAND$ and $NetOR$ networks, and adding two mul^* networks fed with a program $p = [p_1, p_2]$.

I_1	I_2	$Bool_{mul^*}$ with $\mathbf{p}^{NetNAND}$	$NetNAND$	$Bool_{mul^*}$ with $\mathbf{p}^{NetOR}$	$NetOR$
1	1	$\sigma(\bar{y}) \simeq 0.26$	$\sigma(\bar{y}) \simeq 0.08$	$\sigma(\bar{y}) \simeq 1.00$	$\sigma(\bar{y}) \simeq 1.00$
1	0	$\sigma(\bar{y}) \simeq 0.93$	$\sigma(\bar{y}) \simeq 0.92$	$\sigma(\bar{y}) \simeq 0.93$	$\sigma(\bar{y}) \simeq 0.93$
0	1	$\sigma(\bar{y}) \simeq 0.96$	$\sigma(\bar{y}) \simeq 0.93$	$\sigma(\bar{y}) \simeq 0.99$	$\sigma(\bar{y}) \simeq 0.92$
0	0	$\sigma(\bar{y}) \simeq 1.00$	$\sigma(\bar{y}) \simeq 1.00$	$\sigma(\bar{y}) \simeq 0.27$	$\sigma(\bar{y}) \simeq 0.08$

Table 6.6: Output of $NetNAND$ and $NetOR$ networks versus the output of the fixed-weight $Bool_{mul^*}$ fed with the appropriate programming input $\mathbf{p}^{NetNAND}$ and $\mathbf{p}^{NetOR}$. Even in the presence of slightly different numerical values, the *meaning* of the fixed point computation is preserved.

6.4 *Programmer* Network

In this experiment we see a sample deployment of the DMA.

At this purpose we build a network *Programmer* which is capable of sending the right programming inputs to $Bool_{mul^\star}$ DMAN constructed in 6.3 in order to let it be in turn a *nand* and a *or* Boolean function because it interprets the behaviour of $NetNAND$ and $NetOR$ networks. We need *Programmer* to send the right programs $\mathbf{p}^{NetNAND}$ and $\mathbf{p}^{NetOR}$ as output of its attractor computation. In other words we need that *Programmer* is able to approximate the function $f(I)$ assuming the following values:

I	$f(I)$
0	$f(0) = (1, 0)$
1	$f(1) = (0, 1)$

The output of the function will be realized by the fixed points of *Programmer*. To do this our programmer network consists of two neurons, one neuron consisting approximating a *not* Boolean function, the other one realizing an identity function on the values 1 and 0.

The first one is realized by the $NetNOT$ network with a suitable parameter c_2.

$$\dot{y}_2 = -y_2 + b \cdot \sigma(\bar{y}_2) - b \cdot I$$

The other (identity) neuron is cabled by the equation:

$$\dot{y}_1 = -y_1 + w_{11}\sigma(y_1) + I + k_1$$

k_1 is a fixed input on it, and w_{11} is the weight of the auto-connection. The fixed point of this equation is given by

$$\bar{y}_1 = w_{11}\sigma(\bar{y_1}) + I + k_1$$

and we can study the cases

1. $I = 0$ and $\sigma(\bar{y}_1^0) = 0$ implies $\bar{y}_1^0 \approx k_1$
2. $I = 1$ and $\sigma(\bar{y}_1^1) = 1$ implies $\bar{y}_1^1 \approx w_{11} + k_1 + 1$

Choosing $k_1 = -a$, (for example $a = 7$ implies $\sigma(-7) \approx 0.000911$) and consequently for symmetry we can impose an identically (in module) fixed point by setting

$$a = w_{11} + k_1 + 1 = w_{11} - a + 1$$

so that we obtain $w_{11} = 2a - 1$.

In summary the equations of programming will be

$$\dot{y}_1 = -y_1 + (2a - 1) \cdot \sigma(y_1) + I - a$$

$$\dot{y}_2 = -y_2 + b \cdot \sigma(y_2) - b \cdot I$$

Suitably choosing the parameters a and b, we can obtain a *Programmer* network (see Fig. (6.14)) which implements, as good as we want, the behaviour of the function $f(I)$.

6.5 Robustness and time scale problem

CTRNN programmability has been illustrated by assuming a negligible time delay in the stabilization of the multiplication subnetworks activity with respect to the time scale of the original CTRNN network (4.1).

It should be noted that applying the $w-$substitution on a

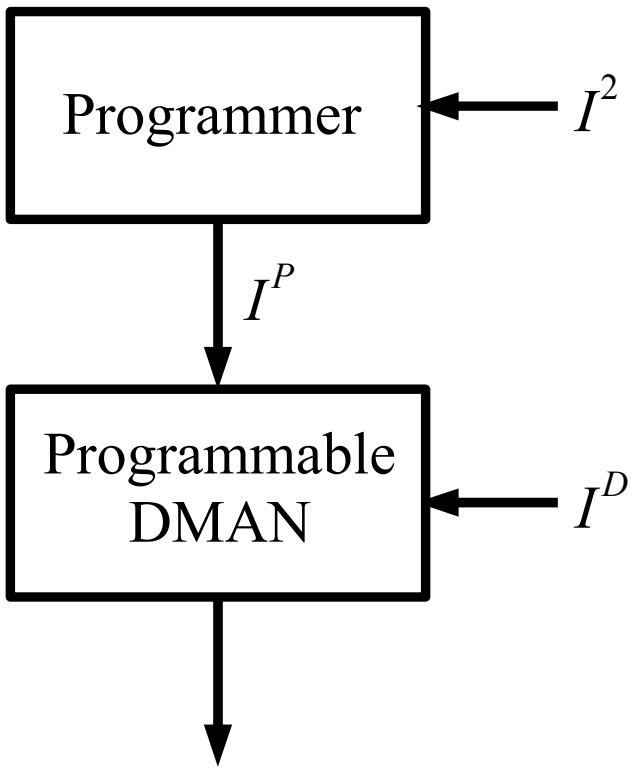

Figure 6.14: DMA Topology. A *Programmer* network is able to send the right program values to a *Programmable* one (e.g. $Bool_{mul^\star}$) showing qualitative different behaviours.

network G we obtain a DMAN G_{mul*} composed of two components: one is the set of the multiplicative networks, the other consists of G's original neurons. The time scales of the two components of the system must differ very significantly if the programming scheme is to succeed. Although a great variability of biological neuron time scales can be hypothesized on the basis of neurophysiologic findings La Camera et al. (2006); Kiebel et al. (2008), in the spirit of biological plausibility_we propose an experimental study to evaluate how changes in the time scales of the two components of the system affect the interpreting capability of our architecture. In particular we test the DMAN robustness under variations of the time constant ratio $r = \tau/\tau^\star$ where τ is the time constant of the neurons belonging to the original network G, and $\tau^\star$ is the time constant of the neurons belonging to the subnetworks $mul^\star$. Indeed the assumption we made of zero or negligible time delay in the stabilization of the virtual weights with respect to the time scale of the original network amounts to treating the virtual weights, as an adiabatic invariant, in mechanics' sense, of the full activity of the w-substituted DMAN $G_{mul^\star}$. In case the time scales of the two components of the system do not differ significantly, i.e., when the ratio r becomes small, the $G_{mul^\star}$ dynamics become much more complex, and our proposed usage of the multiplication subnetworks as providing an interpreting capability might well be jeopardized.

6.6 *NetOne* on different time scales

In the previous case we apply the similarity procedure explained in Chapter 4 to compare networks to which the w-

substitution is being applied. We start from comparing $NetOne$ and $NetOne_{mul^\star}$ when varying the ratio r.

To do this we use the approximated version of sigmoid Pol_M in order to we firstly introduced we consider D^1 as the equation of one neuron with the approximated sigmoid

$$\tau \cdot \frac{dy_1}{dt} = -y_1 + w \cdot Pol_M(y_1) + I$$

with the identity function selectioning observable variable z_1.

$$z_1 = h(y_1) = y_1$$

and as D^2 the substituted one, i.e.

$$\begin{aligned} \tau \cdot \frac{dy_2}{dt} &= -y_2 + x_{mul} + I \\ \tau_{mul} \cdot \frac{dx_{mul}}{dt} &= -x_{mul} + w \cdot Pol_M(y_2) \end{aligned}$$

again with the identity function

$$z_2 = h_2(y_2) = y_2$$

The choice of identity function allows us to compare the values of the potential variables of the two systems y_1 and y_2.

We used as a standard form for the polynomial in order to find all the possibly $\delta-$approximate bisimulation functions for different values of w.

$$V_2(y_1, y_2, x_{mul}) = c_0^1 + c_1^1 y_1 + c_1^2 y_2 + c_1^3 x_{mul} + c_2^1 (y_1)^2 + c_2^2 (y_2)^2 + c_2^3 x_{mul}^2 + c_2^4 y_1 y_2 + c_2^5 y_1 x + c \tag{6.7}$$

We ran three set of experiments on ratio $r_1 = 5$, $r_2 = 50$ and $r_3 = 500$ with $w \in [0, 10]$ and $I = 0$. Of course we expect that

$NetOne$	$\bar{\delta}_{\max}$	standard deviation
$r = 500$	0.0311	$3.9 \cdot 10^{-4}$
$r = 50$	0.2813	0.0117
$r = 5$	5.15	1.65

Table 6.7: Comparison table between different $NetOne$ behaviours with weights $w \in [0, 10]$ and its interpreter in the DMA, showing $\bar{\delta}_{\max}$ which measures the maximum mean bound of $\delta-$approximate bisimulation values. The values are shown for $I = 0$ with all variable initial condition $y_1^0, y_2^0, x_{mul}^0 \in [-1, 1]$ with same initial condition for the output variables $y_1^0 = y_2^0$ to compare.

for high ratio values the equivalence is more accurate. However tests show that for all smaller ratios this degradation for smaller ratios. Table show a significant result. For all parameters values in a range $y_1^0, y_2^0, x_{mul}^0 \in [k_{min}, k_{max}]$, we were always able to find a bisimulation function of the type for any a database of the values randomly drawn from $[0, 10]$. Table 6.7 shows how the mean bound $\bar{\delta}_{\max}$ controls the maximum distance between all possible trajectories between the two systems. Moreover if we took the initial conditions for which $\bar{\delta} = d$ with d a threshold, we could control how many behaviours we can simulate respecting the wanted threshold and how wide is the class of $NetOne$ with different weights the DMAN $NetOne_{mul^\star}$ can $\delta-$approximate bisimulate.

6.7 $NetTwo$ and $NetFive$ cases

Such experiments also test indirectly the robustness of the DMANs with respect to the closeness of the multiplication response function of the mul^* subnetworks to a true product: indeed the value of the multiplication that is returned by the

mul^* subnetworks only becomes acceptable after a setting time of at least some τ .

We conducted two groups of experiments for this study. In the first group, we used a CTRNN network composed of only two nodes, *NetTwo*. The second one involving a network composed of five nodes, *NetFive*. In both cases the networks were numerically integrated and we organized the experiments as follows:

1. We chose N sets of weights and initial conditions for the CTRNN network. Each weight w_{ij} and initial condition $y_i(0)$ has been chosen in a random way into the intervals $[min, max]$ and $[y_{min}, y_{max}]$, respectively. Thus obtaining N networks G^i, with $i \in \{1, \ldots, N\}$.The time constants of the neurons belonging to G^i have been set to τ.

2. We chose three kinds of mul^* subnetworks that differ only in time scales: mul_1^*, mul_2^* and mul_3^* with time constants of the neurons equal to $\tau_1 = \tau/r_1$, $\tau_2 = \tau/r_2$ and $\tau_3 = \tau/r_3$, respectively.

3. For each network G^i we constructed three DMA networks $G^i_{mul*_1}$, $G^i_{mul*_2}$ and $G^i_{mul*_3}$ applying the $w-$*substitution* uniformly by using mul_1^*, mul_2^* and mul_3^*, respectively (see Figure 6.15 in which a depiction of the $w-$substitution for $NetTwo$ is shown).

4. For each ratio $r_k = \tau/\tau_k$ we compared the evolution of the potentials of the networks $G^i_{mul*_k}$ with respect to the networks G^i, with $i \in \{1, \ldots, N\}$: Let us call $\mathbf{y}^i(n)$ and $\tilde{\mathbf{y}}^{i,k}(n)$ the points of the G^i and $G^i_{mul*_k}$ trajectories, restricted at the neurons into G^i, at the time $t = n \cdot \tau$ respectively. Then at each value n we computed the mean

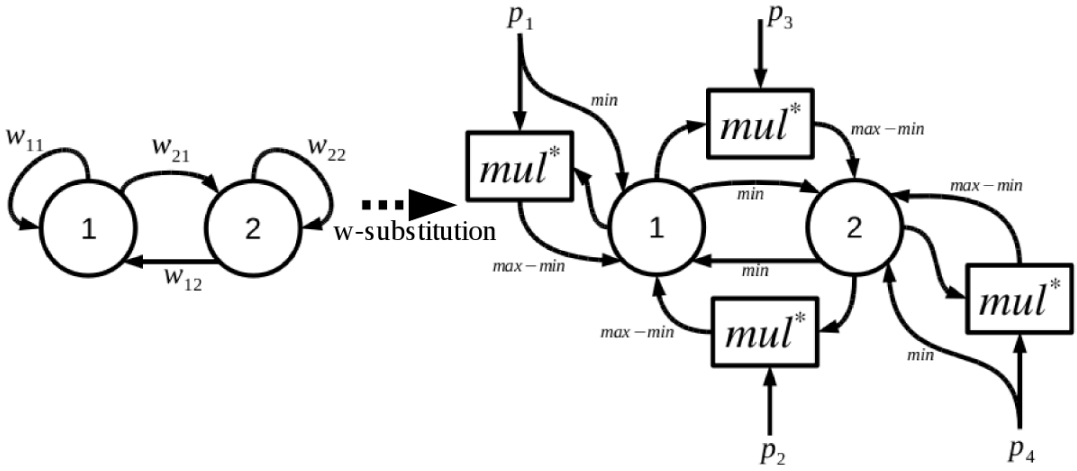

Figure 6.15: $w-$substitution procedure applied to a neural network composed of two neurons.

of the relative euclidean distances

$$\bar{\delta}_n^k = \frac{1}{N}\sum_{i=1}^{N}\frac{\left\|\mathbf{y}^i(n) - \tilde{\mathbf{y}}^{i,k}(n)\right\|}{\|\mathbf{y}^i(n)\|}$$

and the maximum mean bisimulation bound $\bar{\delta}_{\max}$. In this way, the values $\bar{\delta}^k_{\max}$and $\bar{\delta}^k_n$ give a measure of how the behaviours of G and G_{mul*} differ when G_{mul*} is obtained applying uniformly a $w-$*substitution* on G by using in turn the dynamical mul and mul^* subnetworks acting at a time scale which is $1/r_k$ times faster than the time scale of the original network.

5. The parameter used in the two groups of experiments are summarised in Table (6.8)

Parameters	Group 1 Experiments	Group 2 Experiments
M (number of neurons of the original G)	2	5
Number of neurons of G_{mul*}	14	80
min, max (weight interval)	0, 10	0, 10
$\mathbf{y}_{min}$, $\mathbf{y}_{max}$ (initial condition interval)	-30, $+30$	-30, $+30$
$r_1 = \tau/\tau_1^*$	5	5
$r_2 = \tau/\tau_2^*$	50	50
$r_3 = \tau/\tau_3^*$	500	500
N (number of networks)	10^3	10^3

Table 6.8: Experimental Parameters

In Figure 6.16 sample trajectories of $NetTwo$ and $NetTwo_{mul*}$ are shown, restricted to the nodes belonging to $NetTwo$, when the time constant ratio assumes the values r_1, r_2 and r_3. Even in the presence of ratios that make the time constants of mul^* networks very close to the neurons of $NetTwo$, the trajectories are well preserved.

The experimental results show that for each time constant the mean values of the relative distances $\bar{\delta}_k^n$ between the trajectories of the networks $NetTwo$ and $NetTwo_{mul*}$ are initially very high, then they decrease and stabilize at values sufficiently small (less than 0.20%) (see Figure 6.17), probably in correspondence of the presence of fixed points. Accordingly we were always able to find bisimulation functions (see Table 6.9) of the type $V_2(\mathbf{y}_1, \mathbf{y}_2, \mathbf{x}_{mul})$ with which we can control which behaviours we can simulate on the interpreter $NetTwo_{mul^\star}$. Regarding the second set of experiments, in Figure 6.18 sample trajectories and mean values of the relative distances between the trajectory points of $NetFive$ and $NetFive_{mul*}$ are shown. In this case it is possible to note that, although for greater ratios the trajectory similarity is still optimal, smaller ratios

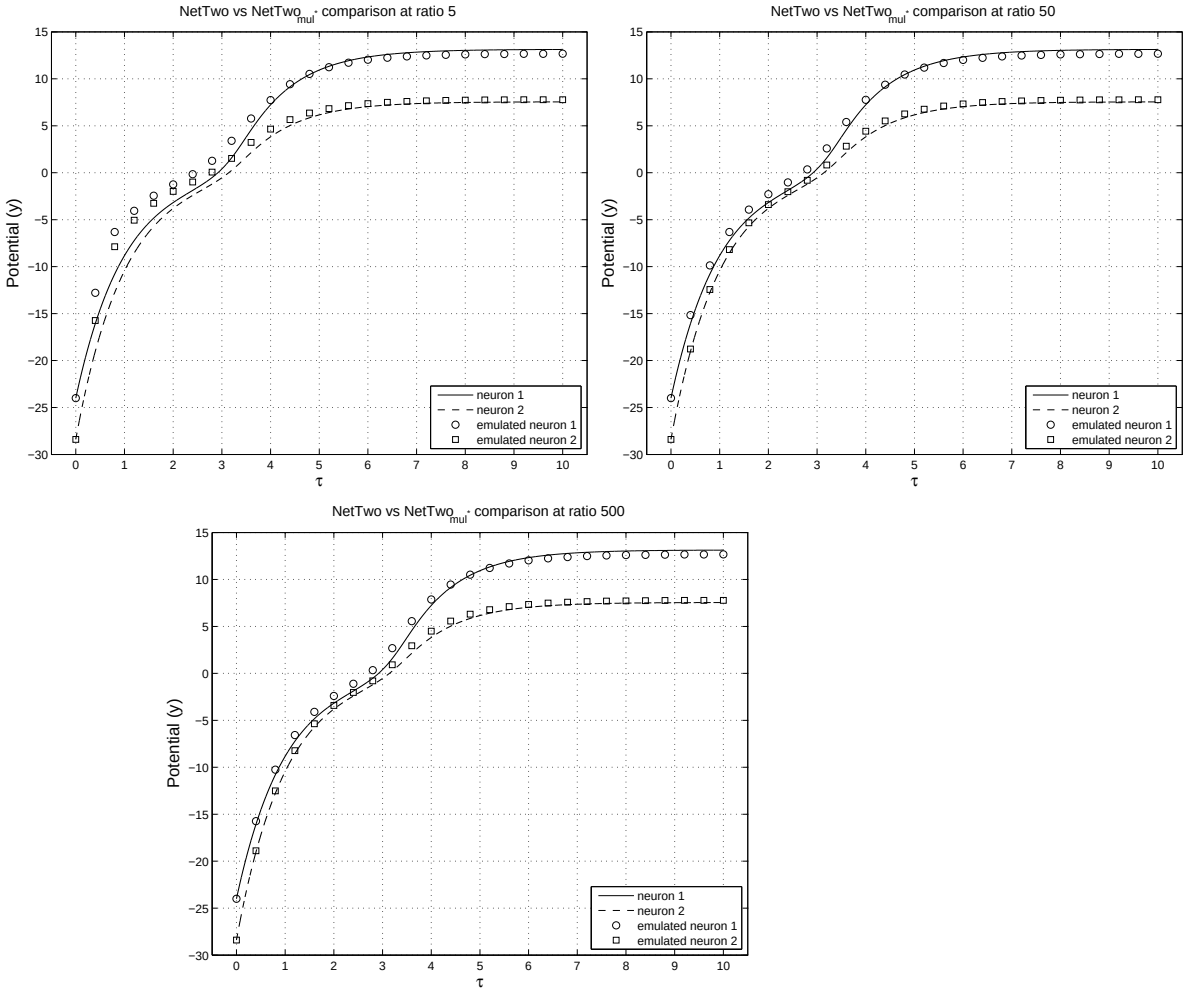

Figure 6.16: Comparison between sample trajectories of $NetTwo$ and $NetTwo_{mul^\star}$. Continuous and dashed lines represent the trajectories of neurons of $NetTwo$. Circles and squares represent trajectories of the corresponding neurons of $NetTwo_{mul^*}$.

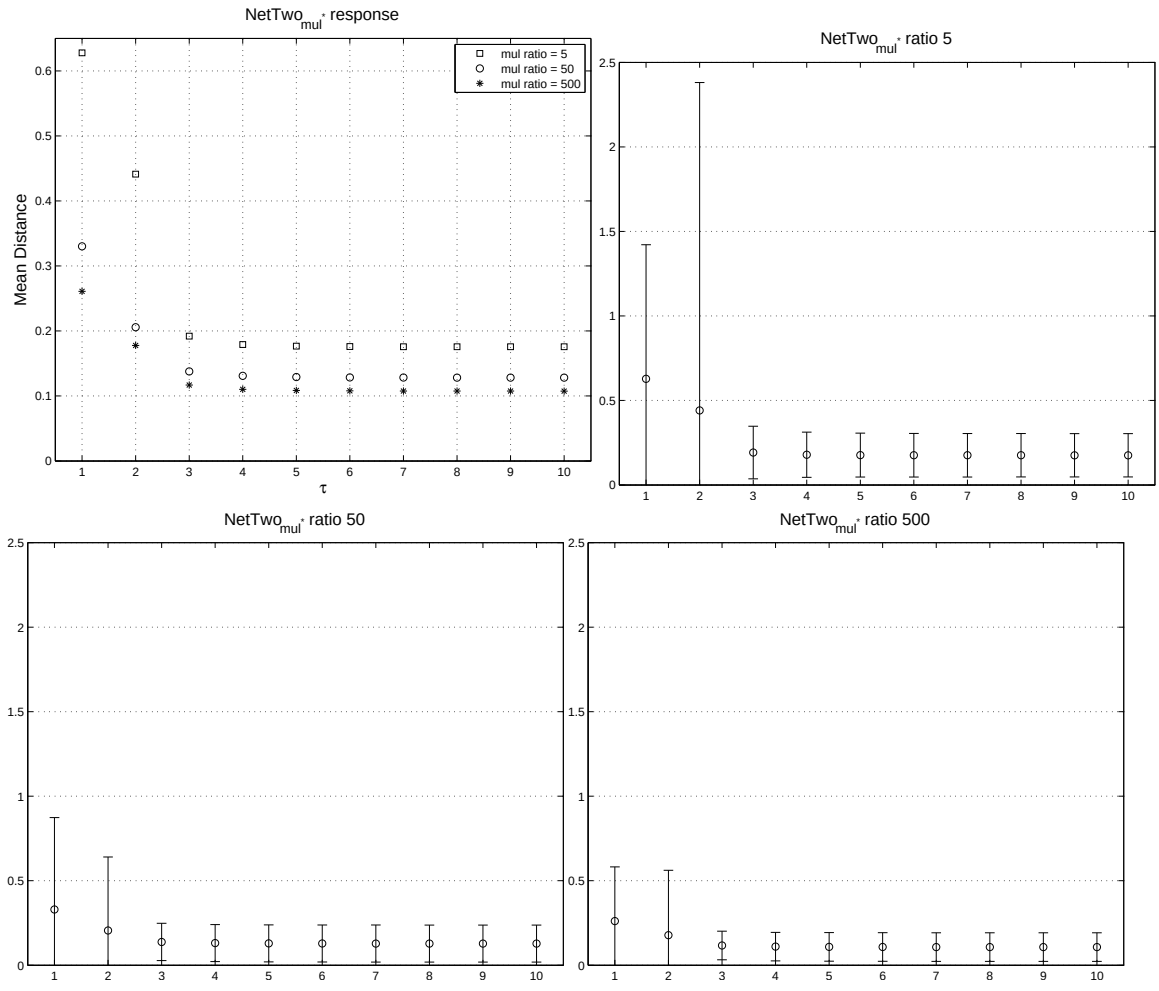

Figure 6.17: Mean and standard deviation of the relative distances between the points of the trajectories of $NetTwo$ and $NetTwo_{mul*}$ at the three different values of the ration r.

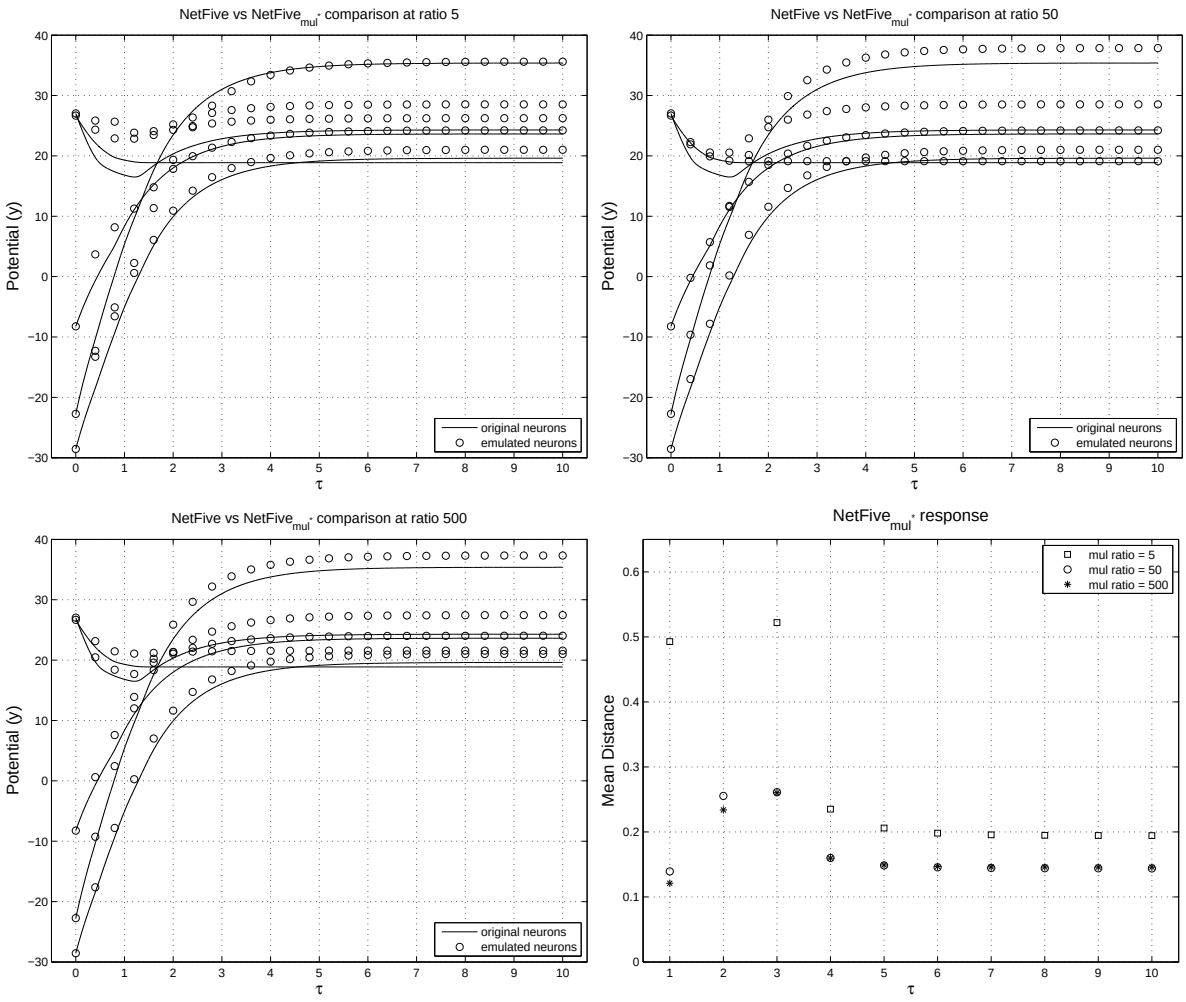

Figure 6.18: Comparison between sample trajectories of $NetFive$ and $NetFive_{mul*}$. Continuous lines represent the trajectories of neurons of $NetFive$. Circles represent the trajectories of the corresponding neurons of $NetFive_{mul*}$. At the bottom right corner the means of the relative distances between the points of the trajectories of $NetFive$ and $NetFive_{mul*}$ at the three different values of the ratio r.

$NetTwo$	$\bar{\delta}$	standard deviation
$r = 500$	5.6913	5.8484
$r = 50$	15.09	16.29
$r = 5$	54.97	50.01

Table 6.9: Comparison table between $NetTwo$ and its correspondent in the DMA, showing $\bar{\delta}$ which measures the maximum mean bound of $\delta-$approximate bisimulation. The values are shown for $I = 0$ with all variable initial condition $\mathbf{y}_1^0, \mathbf{y}_2^0, \mathbf{x}_{mul}^0 \in [-1, 1]$ with same initial condition for the output variables $\mathbf{y}_1^0 = \mathbf{y}_2^0$ to compare.

$NetFive$	$\bar{\delta}$	standard deviation
$r = 500$	23.9024	5.8484
$r = 50$	83.5085	28.81

Table 6.10: Comparison table between $NetFive$ and its correspondent in the DMA, showing $\bar{\delta}$ which measures the maximum mean bound of $\delta-$approximate bisimulation values and the system with the dynamical multiplication. The values are shown for $I = 0$ with all variable initial condition $\mathbf{y}_1^0, \mathbf{y}_2^0, \mathbf{x}_{mul}^0 \in [-1, 1]$ with same initial condition for the output variables $\mathbf{y}_1^0 = \mathbf{y}_2^0$ to compare. For $r = 5$ it is more difficult to find for a large database bisimulation function of the form $V_2(\mathbf{y}_1^0, \mathbf{y}_2^0, \mathbf{x}_{mul}^0)$. Instead for $r = 50$ we succeeded in finding them in the 50% of the cases, and the values in the table refer to them.

could cause the distance between the trajectories of the networks to be appreciable. Accordingly, Table 6.10 shows no problem in finding suitable bisimulation functions for ratio $r = 500$. However for smaller ratio we were not always to find bisimulation function, thus limiting the interpreter capabilities of the architecture when the ratio begin too small. the ability to find bisimulation. However if instead of the potential we consider the output of neurons, which is the real observable variables even in the presence of smaller ratio the similarity

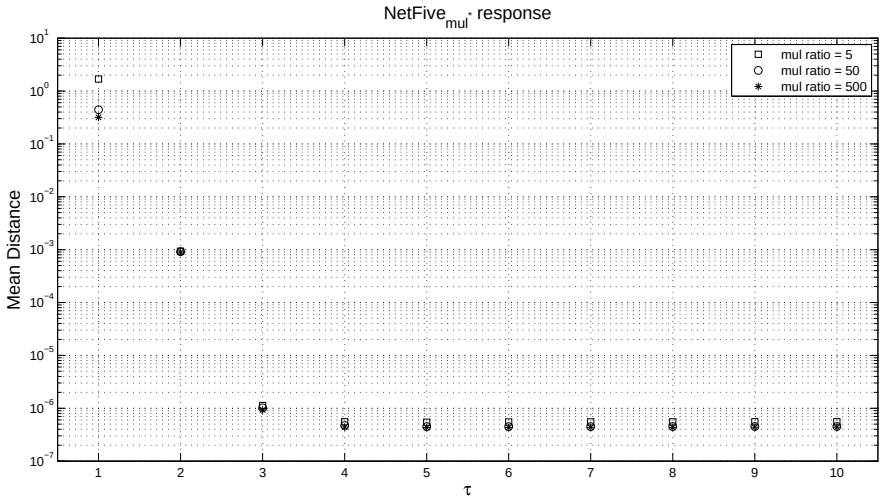

Figure 6.19: Means of the relative distances between the outputs of $NetFive$ and $NetFive_{mul*}$ at the three different values of the ratio r.

seems to be preserved again as the $\bar{\delta}^n$ measures on the output show (see Figure 6.19), suggesting that the recourse at output observable variables could even improve interpreter capability of the architecture.

7 Conclusions and Future Work

7.1 Results of the dissertation

In the thesis dissertation we questioned how biological phenomena controlled by neuronal activity showing properties typically of computational devices could be captured in Artificial Neural Networks.

We have presented a Dynamic Multiplication architecture (DMA), built on a plausible model of biological neuronal networks (CTRNNs). The DMA is conceived in order to show the virtuality capability as defined in Section 3.3 which captures features originally associated with algorithmic computability. Thus DMA allows the building of interpreters of CTRNN Networks, called Dynamic Multiplication Architecture Networks (DMANs) which are special fixed-weight CTRNNs, being programmable because they possess the virtuality capability.

In a plethora of tests on sample networks we have obtained successful implementations and outlined a methodology which might offer biologically plausible modelling of nervous networks endowed with virtuality.

Specifically, in the experiments in Sections 6.2 and 6.6, the

results show that it is possible to obtain a programmable single neuron which, under different programming inputs, exhibits, "on the fly", an appropriate range of qualitatively different behaviours. Since a single neuron with a self-connection is the basic building block for any larger network (Beer, 1995b), these results suggest that virtuality can be built into more complex DMANs.

Moreover experiments in 6.3 show that it is possible to build a small fixed-weight DMAN $Bool_{mul^\star}$ which, under different programming inputs, behaves as the Boolean functions *nand* and *or*. As a consequence, more complex fixed-weight DMANs could be built by composing smaller ones, in order to obtain networks which can be programmed to compute one of a range of Boolean functions. In correspondence we show in experiment 6.4 how a DMAN can be programmed by a *Programmer* Network in order to show different shapes of behaviours at varying programming input on the constructed $Bool_{mul^\star}$.

In Chapter 4, reference was made to the necessity of two different time scales in the operation of the CTRNNs expanded with the multiplicative subnetworks (DMANs). We made the assumptions of zero or negligible time delay in the stabilization of the multiplicative subnetworks with respect to the time scale of the original network. In case the time scales of the two components of the system do not differ significantly, the dynamics might become much more complex and the ensuing behaviour differ sensibly from the intended one. To check this degeneration, in a third set of experiments (Sections 6.5, 6.6 and 6.7) we varied in small networks the ratio of the original network time scale with respect to the mul^* subnetwork time scale. A bisimulation measure δ was given in order to measure the resid-

ual interpreting capabilities of the given DMANs, which allows to control the class which the interpreter is able to simulate. Overall encouraging results have been obtained suggesting that the $w-$substitution is robust with respect to the preservation of the behaviour of the original, not substituted networks.

While the modelling of biological nervous systems usually considers brain areas as special purpose machines and neuronal activity as data, these experiments suggest that the modelling of the activity of some brain areas as multipurpose stored-program machines is a viable hypothesis. In this manner, we propose a clearer notion what computation consists of in biological systems, thus hopefully shedding light on neurophysiological evidences. But a number of open issues must be dealt with before significant modelling and related interaction with the neurobiological research milieu may take place. In this connection we notice two intertwined problematic areas: the relation between learning and virtuality and the implementation of large scale networks through composition of smaller ones.

7.2 Virtuality learning for the DMA

In Section 3.1 we mentioned the fixed-weight line of research which splits into different threads, all related to learning or adaptability, and to which the present work partially belongs.

However, our "fixed-weight" approach is very different from the above threads both in its goal and in its implementation. The programmability capacity, or virtuality, of our DMANs does not resort to any kind of learning or adaptation. Indeed, virtuality is fundamentally different from learning on at least three counts.

First. The time scale of learning is the time scale of synaptic plasticity, usually slower than the time scale of "program switching" which operates at the time scale of the I/O reactivity or, in actual biological networks, at the time scale of the action potential.

Second. The virtuality capability allows an arbitrary (within a certain range) variation of behaviour according to the specifications of the auxiliary (programming) input, while learning occurs as a consequence of the iterated re-enactment of some specific behaviour.

Third. Learning a new behaviour or a new element of information in an ANN generally erases or alters previous memories consigned to the structure of connection weights of the network. In a DMAN the outcome of the learning process is devoted to context data to be assigned to the auxiliary inputs, thus achieving complete independence of the different memory traces. This fact, based on the virtuality capability, might suggest a novel outlook at the open problem of incremental learning.

We observe that with 'programming capability' we did not propose the ability to "write" networks in the same way as one writes programs in some programming language. Rather we wanted to explore the feasibility of *interpretive* ANNs leaving aside for the moment how the specification of the weights which constitute the "program" would be arrived at.

However, we stress that setting the weights as values of the auxiliary inputs is very different from putting them by hand into some platform simulating CTRNNs. Indeed the interpretive architecture of the DMAN carries the setting of the weights

from a structural (in the biological reality: synaptic efficacy) level to a dynamic (in the biological reality: current neuronal activity) level thus allowing the important possibility (see property b. and condition (3) as discussed in Section 3.3 and elsewhere) that the values of the auxiliary or programming inputs could be generated by other segments of a larger network.

In this regard, a considerable open problem is the determination of these values which code the connection weights of a simulated network with some desired behaviour. At the moment, as for $Bool_{mul^*}$ in Section 6.3, our strategy is: firstly, to determine the structure of a CTRNN network G with a specified behaviour either by appropriately training it or, in simple cases, by designing it "by hand"; secondly, to program an interpretive DMAN G_{mul^*} by the programming inputs corresponding to the G structure coming from another *Programmer* network as in Section 6.4. However, in the spirit of biological plausibility, this problem can be splitted into two subproblems.

Where and how to store the programs. That should happen within the larger network so that they will be available when needed. This might be met with some reverberant scheme, but in the end it will certainly require appealing to synaptic plasticity. However there is an important point to stress: in traditional learning it is the structural connection weights that must be altered for memory retention, while in the dynamic multiplication architecture it is the *values* of the connection weights that must be stored.

Learning the programs and recalling them for future use. We have not yet investigated the interesting vistas on

the management of memory which open as a consequence of the interaction between two kinds of learning and training: traditional structural learning vs. learning by storing programs (sets of weights dynamically deployable). It is appealing to reflect on the possibility that "learning by sharing programs" might be a possible embodiment of procedural memory.

7.3 DMANs Compositionality

The composition of generic CTRNNs is the connection of some network outputs to inputs of other networks. In addition to the problems inherent to our asynchronous architecture we have the further difficulty of the requirement of different time-scales for the main and *mul* segments of the network, as discussed in 6.5.

In fact a neural computational structure might well be constructed through the composition of a large number of DMANs by assigning the outputs of the component DMANs to the inputs of other (or the same) DMANs. It is clear that the ensuing structure is still a DMAN, maintaining the fixed-weight feature. Moreover notice that the provided bisimulation measure has the important property that can be obtained by the sum of bisimulation measures given by the composition of networks which interact in a feed forward manner. However when loops are enabled, as the output of some component DMANs may constitute programming inputs to other component DMANs, it follows that the whole DMAN is not just programmable from the outside, but can internally generate programs, thus acquiring the capability of self-programming and code processing at execution time.

Notice that the w-substitution scheme applied to any N-neuron CTRNN allows for the simulation of a no larger than N-neuron network. There are no universal interpreters but only N-neuron networks interpreters for any N. We ran experiments for $N = 2$ and $N = 5$ in Section 6.5. This fact conforms to our distinction between virtuality and universality.

Therefore, a large DMAN composed of interconnected DMANs, although endowed with virtuality, is not bound to a single programmable processor, a CPU in standard programming architectures. Instead many or all component DMANs are programmable units, each within its own range of behaviours, as discussed above, thus obtaining a form of distributed virtuality which is closer implementation-wise to biological nervous tissue than to multiprocessor artificial architectures.

A

Preliminary Mathematical notions

A.1 Topology language

This Appendix contains some of the background nomenclature and terminology that are needed in order to easily read the thesis work. This collection is not meant to be an exaustive mathematical treatment, but is the only the effort to make the thesis work as self-contained as possible.

Definition A.1.1. Let X be a set and T a family of subset of X. T is a *Topology* on X if

- $\emptyset, X \in T$
- $\forall U_i \in T \ \bigcup_{i=1}^{k} U_i \in T$ with a possibly infinite k
- $\forall U_i \in T \ \bigcap_{i=1}^{k} U_i \in T$ with a finite k

Note A.1.2. Topological terminology

- The sets in T are said *open*.
- A subset of X is said to be *closed* if its complement is in T (i.e., it is open).
- A subset of X may be open, closed, both, or neither.

Definition A.1.3. A *compact set* K is a topological space in which for each open cover

$$\{U_h\}_{h\in H} \text{ open} U_h \in X \bigcup_{h\in H} U_h$$

there exists a finite subset $J \subseteq H$, $X = \bigcup_{j\in J} U_j$

Definition A.1.4. The graph of a function $f : U \subseteq \mathbb{R}^n \to \mathbb{R}^m$ is the set

$$G \equiv \{(\mathbf{x}, f(\mathbf{x})) \mid x \in U\} \in \mathbb{R}^{m+n}$$

Definition A.1.5. $f : X \subseteq \mathbb{R}^N \longrightarrow Y \subseteq \mathbb{R}^N$ is a uniformly continuous function if

$$\forall \epsilon > 0 \; \exists \delta > 0 \forall \mathbf{x}_1, \mathbf{x}_2 \in X \text{ such that } \|\mathbf{x}_1 - \mathbf{x}_2\| < \delta \Rightarrow \|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| < \epsilon$$

Definition A.1.6. $C^0(\mathbb{R}^N, \mathbb{R}^N)$ is the set of all the continuous functions $f : \mathbb{R}^N \longrightarrow \mathbb{R}^N$.

Definition A.1.7. $C^k(\mathbb{R}^N, \mathbb{R}^N)$ is the set of all the functions $f : \mathbb{R}^N \longrightarrow \mathbb{R}^N$, k times differentiable and with the $k-th$ derivative $f^k : \mathbb{R}^N \longrightarrow \mathbb{R}^N$ continuous.

Definition A.1.8. A function $f : \mathbb{R}^N \longrightarrow \mathbb{R}^N$ is *smooth* if $f \in C^\infty(\mathbb{R}^N, \mathbb{R}^N)$

Definition A.1.9. A *Homeomorphism* h is a function

$$h : \mathbb{R}^N \longrightarrow \mathbb{R}^N$$

continuous and with a continuous inverse (i.e. $h, h^{-1} \in C^0(\mathbb{R}^N, \mathbb{R}^N)$)

Definition A.1.10. A *Diffeomorphism* h of class C^k is a function

$$h \,:\, \mathbb{R}^N \longrightarrow \mathbb{R}^N$$

if $h, h^{-1} \in C^k(\mathbb{R}^N, \mathbb{R}^N)$

Definition A.1.11. A function $f \,:\, X \subseteq \mathbb{R}^N \longrightarrow Y \subseteq \mathbb{R}^N$ is l*ocally Lipschitz continuous* with respect to $t \in T$ if

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in X \Rightarrow \exists L \geq 0 \parallel f(\mathbf{x}_1) - f(\mathbf{x}_2) \parallel \leq L \parallel \mathbf{x}_1 - \mathbf{x}_2 \parallel$$

Definition A.1.12. A function $f \,:\, X \subseteq \mathbb{R}^N \longrightarrow Y \subseteq \mathbb{R}^N$ is (uniformly) *Lipschitz continuous*

$\exists L \geq 0 \quad$ such that $\quad \forall \mathbf{x}_1, \mathbf{x}_2 \in X \Rightarrow \parallel f(\mathbf{x}_1) - f(\mathbf{x}_2) \parallel \leq L \parallel \mathbf{x}_1 - \mathbf{x}_2 \parallel$

Theorem A.1.13. *If f is uniformly Lipschitz continuous then f is uniformly continuous*

Proof. If f is uniformly Lipschitz continuous then $\exists L \geq 0$ such that $\forall \mathbf{x}_1, \mathbf{x}_2 \in X$ it results $\parallel f(\mathbf{x}_1) - f(\mathbf{x}_2) \parallel \leq L \parallel \mathbf{x}_1 - \mathbf{x}_2 \parallel$. So $\forall \epsilon > 0$, taking $\delta = \epsilon / L$, for all $\parallel \mathbf{x}_1 - \mathbf{x}_2 \parallel < \delta$

$$\parallel f(\mathbf{x}_1) - f(\mathbf{x}_2) \parallel \leq L \parallel \mathbf{x}_1 - \mathbf{x}_2 \parallel < L \cdot \delta = \epsilon$$

□

Proposition A.1.14. *If $f \in C^1(\mathbb{R}^N, \mathbb{R}^N)$ (continuous and differentiable), and f' is bounded then f is Lipschitz continuous*

B

On Turing Virtuality in neural networks

B.1 Turing Virtuality in Neural Networks

In this Thesis we cited different works which assign Turing Universality to ANNs. One of the most cited is within the so called Rational Neural Networks ($\mathbb{Q}$NNs) proved in (Siegelmann and Sontag, 1995; Siegelmann, 1999) where the authors show how to construct a universal neural network of 886 neurons which is capable of simulating a Universal Turing Machine. Thus these systems can be considered programmable insofar as they can simulate any Turing Machine. Their approach differs significantly in goals and motivation from the one in this thesis. They succeeded in simulating inside the network model they chose the Turing Universality or as we called it Turing Virtualiy. On the other hand inside the thesis we searched for Material Virtuality, programmability which is not related to third devices (Turing machines) but is within the model itself.

In this appendix we review in details the proof of Siegelmann and Sontag, thus clarifying how this approach is different from ours.

B.2 Rational Neural Network Model

We briefly describe the Neural Network Model used for the demonstration. We consider N neurons updated by the law

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} w_i^j x_j + \sum_{h=1}^{M} b_i^h I_h + C_i\right) \tag{B.1}$$

where

- x_i terms are the activations of the neurons
- w_{ij} terms are the weights connecting the neurons of the net
- I_h terms are external inputs to the net
- b_i^h terms are weights to the external inputs
- C_i terms are bias constants
- σ_S is the saturated sigmoid defined as

$$\sigma_S(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$

In this way if the inputs and the parameters of the nets are rational numbers also the neuron values will be rational. In this sense we referred to this model as a *Rational Neural Network* ($\mathbb{Q}$NN). Notice that this model is a discrete-time model.

Considering the state this system could reach, a network can be seen as a functional with M inputs

$$f : (\mathbf{x}, I_1, \ldots, I_M) \in \mathbb{Q}^N \times \{0,1\}^M \longrightarrow \mathbb{Q}^N$$

However in the construction of the net we will use a "reduced" version

$$f : (\mathbf{x}, I^D, I^V)\mathbb{Q}^N \times \{0,1\}^2 \longrightarrow \mathbb{Q}^N$$

and so the equation (B.1) will be

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} w_i^j x_j + I^D(t) + I^V(t)\right)$$

The entire state of the network can be seen as a vector of N neurons

$$\mathbf{x} = (x_1, \ldots, x_N)$$

The evolution of the network state in time will be denoted with a superscript value: for example the state of the network at time t will be denoted with $\mathbf{x}^t$.

Actually in these notes we will consider a net *without* inputs that starting from an initial state is able to execute every computation of a Turing Machine. It would be possible to add two inputs to this network in order to let them reach the desired initial state to start the appropriate computation, anyway this construction will not be analyzed here.

B.2.1 4−Cantor-like Encoding

The particular encoding of the state of the neurons provided in the architecure is given by:

Definition B.2.1. 4-Cantor-like encoding δ_4 of binary strings on an alphabet $\Sigma = \{0,1\}$ allows the encoding of stack values. It is defined as

$$\delta_4 : \Sigma^* \longrightarrow \mathbb{Q}$$

so that

- $\delta_4(\epsilon) = 0$
- $\delta_4(\alpha = \alpha_1, \ldots, \alpha_k) = \sum_{i=1}^{|\alpha|} \frac{2\alpha_i+1}{4^i}$

where ϵ is the empty string,$\alpha_i \in \Sigma$ and $\alpha \in \Sigma^*$.

This encoding allows to write binary strings (e.g. $\alpha = 1001_2$) as rational numbers (e.g. $\delta_4(\alpha) = \frac{3}{4} + \frac{1}{16} + \frac{1}{64} + \frac{3}{256}$). Moreover it has the advantage of creating *numerical gaps* (see Fig. (B.1)) which will be used during the construction of the p-stack Rational Neural Network machine. A string with only one 0 will be encoded as 1/4, while a string with only one 1 will be encoded as 3/4.

If we consider that the series convergence

$$\delta_4(\alpha) = \sum_{i=1}^{|\alpha|} \frac{2\alpha_i + 1}{4^i} \leq 3 \cdot \sum_{i=1}^{\infty} \frac{1}{4^i} = 3 \cdot \left(\frac{1}{1 - 1/4} - 1 \right) = 3 \cdot \frac{1}{3} = 1$$

it is clear that the values of the encoding will be bounded between 0 (the empty string) and 1 (a string with infinite 1). Moreover, for each string α_0 the first symbol of which is 0 we obtain

$$\delta_4(\alpha_0) \leq \frac{1}{4} + 3 \cdot \sum_{i=2} \frac{1}{4^i} = \frac{1}{4} + 3 \cdot \left(\frac{1}{1 - 1/4} - 1 - \frac{1}{4} \right) = \frac{1}{2}$$

Thus the strings beginning with 0 can assume values in the interval $[\frac{1}{4}, \frac{1}{2})$ with $\frac{1}{2}$ as the code of the string starting with the symbol 0 and followed by an infinite sequence of 1. Analogously with a string α_1 starting with the symbol 1 we obtain

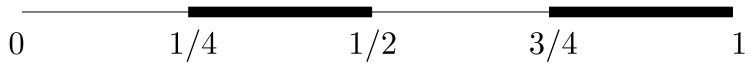

Figure B.1: 4−Cantor-like encoding of binary strings

$$\delta_4(\alpha_1) \leq \frac{3}{4} + 3 \cdot \sum_{i=2} \frac{1}{4^i} = \frac{3}{4} + 3 \cdot \left(\frac{1}{1 - 1/4} - 1 - \frac{1}{4} \right) = 1$$

meaning that strings starting with 1 can assume values in the interval $[\frac{3}{4}, 1)$. These properties will be at the basis of the definition of the operations in the paragraph (B.4.1).

B.2.2 Equivalence between a Turing Machine and a $\mathbb{Q}$NN

As a $p-$stack machine can *simulate* a Turing machine, it will be sufficient to prove the following theorem:

Theorem B.2.2. *(Siegelmann and Sontag) [Equivalence between a $p-$ stack and a Rational Neural Network] Given a function ψ*

$$\psi : \{0,1\}^+ \longrightarrow \{0,1\}^+$$

computed by a $p-$stack Turing machine

$$T : \mathbb{N} \longrightarrow \mathbb{N}$$

there exists a Rational Neural Network R which computes $\psi(\omega)$ with $\omega \in \{0,1\}^+$ so that given the initial network state

$$\mathbf{x}^0(\omega) = (\delta_4(\omega), 1, 0, \ldots, 0)$$

the computation made by R is:

- *undefined if* $\forall j\ x_3^j = 0$
- *defined if* $\exists r\ :\ \forall j \in \{0, \ldots, r-1\} x_3^j = 0 \wedge x_3^r = 1$ *and the result is* $x_1^r = \delta_4(\psi(\omega))$*, so the network state at time* r*, when the computation is over, will be something like:*

$$\mathbf{x}^r(\omega) = (\psi(\omega), ?, 1, \ldots, ?)$$

This theorem summarizes the result of the work, which is that a $\mathbb{Q}NN$ can simulate a $p-$stack machine.

B.3 $p-$stack machine

In this section we briefly explain how a $p-$stack machine works, before constructing a network respecting the model (B.1) which will be able to simulate it.

A stack machine can do four operations on a binary valued stack:

- `no` $-$ `op`, which means to leave the stack unaltered
- `push`0, which means to add a 0 on the top of the stack
- `push1`, which means to add a 1 on the top of the stack
- `pop`, which means to delete the first element of the stack

Moreover it is possible to read the top of the stack by an operation

$$\texttt{top}\ :\ stack_h \longrightarrow \texttt{top}(stack_h) = \alpha_h \in \{0, 1\}$$

and to control if the stack is empty by an operation

$$\texttt{non} - \texttt{empty}\ :\ stack_h \longrightarrow \texttt{non} - \texttt{empty}(stack_h) = \alpha_h \in \{0, 1\}$$

which returns 1 if the stack is $\texttt{non}-\texttt{empty}$ and 0 otherwise.

Given p stack structures, a $p-$stack machine M can be defined as $(p+4)$-tuple

$$M \stackrel{def}{=} (Q, q_I, q_H, \theta_0, \theta_1, \ldots, \theta_p)$$

where

- Q is a finite set of states
- $q_I \in Q$ is the initial state
- $q_H \in Q$ is the halting (final) state
- θ_0 is the next state map

  $$\theta_0 \, : \, Q \times \{0,1\}^{2p} \longrightarrow Q$$

  which maps the current state, the first elements of the stacks $(\texttt{top}_1, \ldots, \texttt{top}_p \in \{0,1\}^p)$, if present, and $(\texttt{non}-\texttt{empty}_1, \ldots, \texttt{non}-\texttt{empty}_p \in \{0,1\}^p)$ into the next state the machine will reach.

- θ_h for $h \in \{1, \ldots, p\}$ taking information from stacks similarly as explained before

  $$\theta_h \, : \, Q \times \{0,1\}^{2p} \longrightarrow \{\texttt{no}-\texttt{op}, \texttt{push0}, \texttt{push1}, \texttt{pop}\}$$

An instant configuration of a $p-$stack machine is a $(p+1)$-tuple

$$(q, stack_1, \ldots stack_p)$$

with $q \in Q$ and $stack_h$ the information in the stack.

B.3.1 2−stack copy machine

As a running example we will show a simple 2−stack machine which copies the string present at the beginning of the computation in the first stack, reverted on the second stack:

$$Copy \stackrel{def}{=} (Q, q_I, q_H, \theta_0, \theta_1, \ldots, \theta_p)$$

where

- $Q = \{q_0, q_1, q_2\}$
- $q_I = q_0$
- $q_H = q_2$
- θ_0 is defined by the Table (B.1)

q	$\mathtt{top}_1$	$\mathtt{top}_2$	$\mathtt{non-empty}_1$	$\mathtt{non-empty}_2$	θ_0
q_0	t_1	t_2	1	e_2	q_1
q_0	t_1	t_2	0	e_2	q_2
q_1	t_1	t_2	e_1	e_2	q_0

Table B.1: Next state transition

where $t_1, t_2, e_1$and e_2 could assume any possible values.

- θ_1 is defined in Table (B.2)

q	$\mathtt{top}_1$	$\mathtt{top}_2$	$\mathtt{non-empty}_1$	$\mathtt{non-empty}_2$	θ_1
q_0	t_1	t_2	e_1	e_2	$\mathtt{no-op}$
q_1	t_1	t_2	e_1	e_2	$\mathtt{pop}$

Table B.2: Stack 1 transition

- θ_2 is defined in Table (B.3)

q	$\texttt{top}_1$	$\texttt{top}_2$	$\texttt{non}-\texttt{empty}_1$	$\texttt{non}-\texttt{empty}_2$	θ_2
q_0	0	t_2	1	e_2	push0
q_0	1	t_2	1	e_2	push1
q_0	t_1	t_2	0	e_2	$\texttt{no}-\texttt{op}$
q_1	t_1	t_2	e_1	e_2	$\texttt{no}-\texttt{op}$

Table B.3: Stack 2 transition

It is self-evident how an execution of this machine will reach its goal. For example, given a string on the first stack $\alpha = 001$ and the other stack empty, the step by step execution would be

1. State of the machine: $stack_1 = (0, 0, 1)$ $stack_2 = ()$, initial state q_0 so $State = (q_0, stack_1, stack_2)$. Applying $\theta_1(q_0, 0, 0, 1, 0)$ and $\theta_2(q_0, 0, 0, 1, 0)$, 0 is pushed into the second stack. Applying $\theta_0(q_0, 0, 0, 1, 0)$ the next state will be q_1.

2. State of the machine: $stack_1 = (0, 0, 1)$ $stack_2 = (0)$, state q_1 so $State = (q_1, stack_1, stack_2)$. Applying $\theta_1(q_1, 0, 0, 1, 1)$ and $\theta_2(q_1, 0, 0, 1, 1)$, 0 is popped from the first stack. Applying $\theta_0(q_1, 0, 0, 1, 1)$ the next state will be q_0.

3. State of the machine: $stack_1 = (0, 1)$ $stack_2 = (0)$, state q_0 so $State = (q_0, stack_1, stack_2)$. Applying $\theta_1(q_0, 0, 0, 1, 1)$ and $\theta_2(q_0, 0, 0, 1, 1)$, 0 is pushed into the second stack. Applying $\theta_0(q_0, 0, 0, 1, 1)$ the next state will be q_1.

4. State of the machine: $stack_1 = (0, 1)$ $stack_2 = (0, 0)$, state q_1 so $State = (q_1, stack_1, stack_2)$. Applying $\theta_1(q_1, 0, 0, 1, 1)$

and $\theta_2(q_1, 0, 0, 1, 1)$, 0 is popped from the first stack. Applying $\theta_0(q_1, 0, 0, 1, 1)$ the next state will be q_0.

5. State of the machine: $stack_1 = (1)$ $stack_2 = (0, 0)$, state q_0 so $State = (q_0, stack_1, stack_2)$. Applying $\theta_1(q_0, 1, 0, 1, 1)$ and $\theta_2(q_0, 1, 0, 1, 1)$, 1 is pushed into the second stack. Applying $\theta_0(q_0, 1, 0, 1, 1)$ the next state will be q_1.

6. State of the machine: $stack_1 = (1)$ $stack_2 = (1, 0, 0)$, state q_1 so $State = (q_1, stack_1, stack_2)$. Applying $\theta_1(q_1, 1, 0, 1, 1)$ and $\theta_2(q_1, 1, 0, 1, 1)$, 1 is popped from the first stack. Applying $\theta_0(q_1, 1, 0, 1, 1)$ the next state will be q_0.

7. State of the machine: $stack_1 = ()$ $stack_2 = (1, 0, 0)$, state q_0 so $State = (q_0, stack_1, stack_2)$. Applying $\theta_1(q_0, 0, 0, 0, 1)$ and $\theta_2(q_0, 0, 0, 1, 1)$, no operation is made on the stacks. Applying $\theta_0(q_0, 0, 0, 0, 1)$ the next state will be the halting state q_2.

B.4 Rational Neural Network Construction

This section constructs the pieces of network to perform the operation of the $p-$machine:

- the stack operations,
- the update of the states and the stacks

B.4.1 Simulation of Stack Operations

Given a binary string α (e.g. 0100_2), and its encoding $g = \delta_4(\alpha)$ (e.g. $g = \frac{1}{4} + \frac{3}{4^2} + \frac{1}{4^3} + \frac{1}{4^4}$), we can define "clever" *sigma* operations that allow us to construct nets *simulating* the operation on stacks defined in (B.4.1):

- $\texttt{top}(g) = \sigma(4g - 2)$. In fact

$$\sigma(4g-2) = \begin{cases} 0 & \text{if } g \in [1/4, 1/2] \\ 1 & \text{if } g \in [3/4, 1] \end{cases}$$

- $\texttt{push0}(g) = \sigma(\frac{g}{4} + \frac{1}{4})$. This operation is simply obtained by dividing all the encoding by 4 and adding the first bit encoding 1/4

- $\texttt{push1}(g) = \sigma(\frac{g}{4} + \frac{3}{4})$. This operation is simply obtained by dividing all the encoding by 4 and adding the first bit encoding 3/4

- $\texttt{pop}(g) = \sigma(4g - (2\texttt{top}(g) + 1))$. This operation is obtained by subtracting the first element of the stack and by multiplying the result by 4

- $\texttt{non} - \texttt{empty}(g) = \sigma(4g)$. In fact

$$\sigma(4g) = \begin{cases} 0 & \text{if } g = 0 \\ 1 & \text{if } g \geq 1/4 \end{cases}$$

B.4.2 Neural $p-$Stack Machine

Let us construct the dynamical system N that will simulate a $p-$stack machine $M = (Q = \{q_1, \ldots, q_s\}, q_I, q_H, \theta_0, \theta_1, \ldots, \theta_p)$

$$N : (x_1, \ldots, x_s, g_1, \ldots, g_p) \in \mathbb{Q}^{s+p} \longrightarrow N(x_1, \ldots, x_s, g_1, \ldots, g_p) \in \mathbb{Q}^{s+p}$$

The first step is the encoding of the machine state q_i on a subset of neurons of the network N.

The activity of the first s neurons $x_1, \ldots, x_s$ codifies the state of the machine as showed in Table (B.4):

	x_1	x_2	$\ldots$	x_i	$\ldots$	x_s
q_1	1	0	$\ldots$	0	$\ldots$	0
q_2	0	1	0	$\ldots$	$\ldots$	0
$\vdots$	$\vdots$	$\ddots$		$\ddots$		$\vdots$
q_i	0	$\ldots$	0	1	$\ddots$	0
$\vdots$	$\vdots$			$\ddots$		$\vdots$
q_s	0	$\ldots$	0	$\ldots$	0	1

Table B.4: First s neurons state encoding

The second step is to find functions of the type:

$$f : \{0,1\}^{2p} \longrightarrow \{0,1\}$$

Given $\mathbf{read} = (\mathtt{top}(g_1), \ldots \mathtt{top}(g_h), \mathtt{non{-}empty}(g_1), \ldots, \mathtt{non{-}empty}(g_h))$ we define

- β_{ij} as

$$\beta_{ij}(\mathbf{read}) = \begin{cases} 1 & \text{if } \theta_0(j, \mathbf{read}) = i \\ 0 & \text{otherwise} \end{cases} \tag{B.2}$$

  with $i, j \in \{1, \ldots, s\}$

- γ^1_{hj} as

$$\gamma^1_{hj}(\mathbf{read}) = \begin{cases} 1 & \text{if } \theta_h(j, \mathbf{read}) = \mathtt{no-op} \\ 0 & \text{otherwise} \end{cases} \tag{B.3}$$

  with $j \in \{1, \ldots, s\}$ e $h \in \{1, \ldots, p\}$

- γ^2_{hj} as

$$\gamma^2_{hj}(\mathbf{read}) = \begin{cases} 1 & \text{if } \theta_h(j, \mathbf{read}) = \texttt{push0} \\ 0 & \text{otherwise} \end{cases} \tag{B.4}$$

  with $j \in \{1, \ldots, s\}$ e $h \in \{1, \ldots, p\}$

- γ^3_{hj} as

$$\gamma^3_{hj}(\mathbf{read}) = \begin{cases} 1 & \text{if } \theta_h(j, \mathbf{read}) = \texttt{push1} \\ 0 & \text{otherwise} \end{cases} \tag{B.5}$$

  with $j \in \{1, \ldots, s\}$ and $h \in \{1, \ldots, p\}$

- γ^4_{hj} as

$$\gamma^4_{hj}(\mathbf{read}) = \begin{cases} 1 & \text{if } \theta_h(j, \mathbf{read}) = \texttt{pop} \\ 0 & \text{otherwise} \end{cases} \tag{B.6}$$

  with $j \in \{1, \ldots, s\}$ and $h \in \{1, \ldots, p\}$

The update of the "state" neurons can be written as

$$x_i^{t+1} = \sum_{j=1}^{s} \beta_{ij}(\mathbf{read}^t) x_j^t \tag{B.7}$$

and the update of the "stack" neurons as

$$\begin{array}{rl} g_h^{t+1} = & \sum_{j=1}^{s} \gamma^1_{hj}(\mathbf{read}^t) \cdot g_h^t + \\ & \sum_{j=1}^{s} \gamma^2_{hj}(\mathbf{read}^t) \cdot \texttt{push0}(g_h^t) \\ & \sum_{j=1}^{s} \gamma^3_{hj}(\mathbf{read}^t) \cdot \texttt{push1}(g_h^t) \\ & \sum_{j=1}^{s} \gamma^4_{hj}(\mathbf{read}^t) \cdot \texttt{pop}(g_h^t) \end{array} \tag{B.8}$$

This transitions exactly simulate the $p-$stack machine. The

only problem left is to find "sigma functions" for (B.2), (B.3), (B.4), (B.5) and (B.6) in order to achieve this part of the computation by suitable subnetworks.

2−stack CopyNet Machine

Considering the machine *Copy* defined in (B.3.1) we start building a subnet with 5 neurons: x_1, x_2, x_3, g_1 and g_2. The first 3 neurons encode the states as explained in Table (B.4)

	x_1	x_2	x_3
q_1	1	0	0
q_2	0	1	0
q_3	0	0	1

Table B.5: *Copy* Machine state encoding

The appropriate strings with the 4−Cantor like encoding described in (B.2.1) will be encoded on neurons g_1 (first stack) and g_2 (second stack).

The transitions from a state to another is made by the function $\beta_{ij}(\mathbf{read})$ that can be deducted by (B.1), (B.2) and (B.1).

	$\mathtt{top}_1$	$\mathtt{top}_2$	$\mathtt{non-empty}_1$	$\mathtt{non-empty}_2$	$\beta_{ij}(\mathbf{read})$
β_{11}	t_1	t_2	e_1	e_2	0
β_{12}	t_1	t_2	1	e_2	1
β_{12}	t_1	t_2	0	e_2	0
β_{21}	t_1	t_2	e_1	e_2	1
β_{22}	t_1	t_2	e_1	e_2	0
β_{13}	t_1	t_2	1	e_2	0
β_{13}	t_1	t_2	0	e_2	1
β_{23}	t_1	t_2	e_1	e_2	0
β_{31}	t_1	t_2	e_1	e_2	0
β_{32}	t_1	t_2	e_1	e_2	0
β_{33}	t_1	t_2	e_1	e_2	0

Table B.6: *Copy* Machine $\beta_{ij}(\mathbf{read})$ transition

The table is completely analogous to Table (B.1), the only difference is that here also the non-transitions are depicted: for example for β_{11} there are no possible transitions (always 0 values whatever values **read** assumes). If we assume that the network is in the state $(x_1^t, x_2^t, x_3^t) = (1, 0, 0)$ and $g_1 = 1/4$ (that is a 0 on the first stack) and $g_2 = 0$ (the empty string on the second)

$$x_1^{t+1} = \beta_{11}x_1^t + \beta_{12}x_2^t + \beta_{13}x_3^t = 0$$

$$x_2^{t+1} = \beta_{21}x_1^t + \beta_{22}x_2^t + \beta_{23}x_3^t = 1$$

$$x_3^{t+1} = \beta_{31}x_1^t + \beta_{32}x_2^t + \beta_{33}x_3^t = 0$$

So the next state is $(x_1^{t+1}, x_2^{t+1}, x_3^{t+1}) = (0, 1, 0)$.

Similarly from the definitions (B.3), (B.4), (B.5) and (B.6) and the Tables (B.2) and (B.3) we can construct the relative ta-

bles for the γ^k_{hj} functions with $k \in \{1,2,3,4\}$ (the operations), $h \in \{1,2\}$ (the stacks) $j \in \{1,2,3\}$ (the states).

	$\texttt{top}_1$	$\texttt{top}_2$	$\texttt{non}-\texttt{empty}_1$	$\texttt{non}-\texttt{empty}_2$	$\gamma^1_{hj}(\textbf{read})$
γ^1_{11}	t_1	t_2	e_1	e_2	0
γ^1_{12}	t_1	t_2	e_1	e_2	1
γ^1_{21}	t_1	t_2	0	e_2	1
γ^1_{21}	t_1	t_2	1	e_2	0
γ^1_{22}	t_1	t_2	e_1	e_2	1

Table B.7: *Copy* Machine $\gamma^1_{ij}(\textbf{read})$ - $\texttt{no}-\texttt{op}$ transition

	$\texttt{top}_1$	$\texttt{top}_2$	$\texttt{non}-\texttt{empty}_1$	$\texttt{non}-\texttt{empty}_2$	$\gamma^2_{hj}(\textbf{read})$
γ^2_{11}	t_1	t_2	e_1	e_2	0
γ^2_{12}	t_1	t_2	e_1	e_2	0
γ^2_{21}	0	t_2	1	e_2	1
γ^2_{21}	1	t_2	1	e_2	0
γ^2_{21}	t_1	t_2	0	e_2	0
γ^2_{22}	t_1	t_2	e_1	e_2	0

Table B.8: *Copy* Machine $\gamma^2_{ij}(\textbf{read})$ - $\texttt{push}0$ transition

	$\texttt{top}_1$	$\texttt{top}_2$	$\texttt{non}-\texttt{empty}_1$	$\texttt{non}-\texttt{empty}_2$	$\gamma^3_{hj}(\textbf{read})$
γ^3_{11}	t_1	t_2	e_1	e_2	0
γ^3_{12}	t_1	t_2	e_1	e_2	0
γ^3_{21}	1	t_2	1	e_2	1
γ^3_{21}	0	t_2	1	e_2	0
γ^3_{21}	t_1	t_2	0	e_2	0
γ^3_{22}	t_1	t_2	e_1	e_2	0

Table B.9: *Copy* Machine $\gamma^3_{ij}(\textbf{read})$ - $\texttt{push}1$ transition

	$\texttt{top}_1$	$\texttt{top}_2$	$\texttt{non}-\texttt{empty}_1$	$\texttt{non}-\texttt{empty}_2$	$\gamma^4_{hj}(\mathbf{read})$
γ^4_{11}	t_1	t_2	e_1	e_2	0
γ^4_{12}	t_1	t_2	e_1	e_2	1
γ^4_{21}	t_1	t_2	e_1	e_2	0
γ^4_{22}	t_1	t_2	e_1	e_2	0

Table B.10: *Copy* Machine $\gamma^4_{ij}(\mathbf{read})$ - pop transition

B.4.3 Decomposition Theorem

How do we know that it is always possible to build γ and β functions respecting the given $p-$stack machine? This section give us the wanted results:

Lemma B.4.1. *For each $l_1, l_2, \ldots, l_k \in \{0,1\}$ it is possible to write*

$$l_1 \cdot l_2 \ldots l_k = \sigma(l_1 + l_2 + \cdots + l_k - k + 1)$$

Theorem B.4.2. *$\forall f : \{0,1\}^t \longrightarrow \{0,1\}$ $\exists \mathbf{v}_1, \ldots, \mathbf{v}_{2^t} \in \mathbb{Z}^{t+2}$ and $c_1, \ldots, c_{2^t} \in \mathbb{Z}$ such that $\forall s_1, \ldots, s_t x \in \{0,1\}$ and $g \in [0,1)$ it is possible to have*

$$f(s_1, \ldots, s_t)xg = \sigma(g + \sum_{r=1}^{2^t} c_r \sigma(\mathbf{v}_r \cdot \mathbf{h}) - 1)$$

where $\mathbf{h} = (1, s_1, \ldots, s_t, x)$

Proof. It is possible to expand $f(s_1, \ldots, s_t)$ in polynomial:

$$f(s_1, \ldots, s_t) = c_1 + c_2 s_1 + \cdots + c_{t+1} s_t + c_{t+2} s_1 s_2 + \cdots + c_{2^t} s_1 s_2 \ldots s_t \quad \text{(B.9)}$$

Multiplying by x, using the fact that $x = \sigma(x)$ and the Lemma (B.4.1) we can write

$$\begin{aligned} f(s_1,\ldots,s_t)x &= c_1\sigma(x) + c_2\sigma(x + s_1 - 2 + 1) + \cdots + c_{2^t}\sigma(s_1 + s_2 + \cdots + s_t + x \\ &= \sum_1^{2^t} c_r\sigma(\mathbf{v}_r \cdot \mathbf{h}) \end{aligned}$$

Finally reapplying Lemma (B.4.1) to $f(s_1,\ldots,s_t)xg$ (where $l_1 = f(s_1,\ldots,s_t)x$ and $l_2 = g$), we obtain the desired proof of the Theorem (B.4.2)

$$\begin{aligned} f(s_1,\ldots,s_t)xg &= \sigma(g + f(s_1,\ldots,s_t)x - 2 + 1) = \\ &= \sigma(g + \sum_1^{2^t} c_r\sigma(\mathbf{v}_r \cdot \mathbf{h}) - 1) \end{aligned}$$

□

B.4.4 Function Construction in 3 layers

Now in the formulation of the Theorem (B.4.2) substituting g respectively with 1, g_h, $\texttt{push0}(g_h)$, $\texttt{push1}(g_h)$ and $,\texttt{pop}(g_h)$ we realize how to construct nets computing β_{ij}, γ^1_{hj}, γ^2_{hj}, γ^3_{hj} and γ^4_{hj}.

In fact firstly it is possible to construct a layer L^3 of $s \cdot 2^{2p}$ $1-$neuron subnetworks $\mathcal{L}^r : \mathbb{N}^{2p+2} \longrightarrow \mathbb{N}$ which computes

$$o^j_r(\mathbf{read}, x_j) = \sigma(v^0_r + v^1_r x_1 + \cdots + v^p_r x_p + v^{p+1}_r g_1 \cdots + v^{2p}_r g_p + v^{2p+1}_r x_j)$$

and then we can construct

- all the β_{ij} for $i \in \{1,\ldots,s\}$ with s $1-$neuron nets $\mathcal{L}^{\beta_i}$:

$\mathbb{N}^{2^{2p}} \longrightarrow \mathbb{N}$

$$o^{\beta_j} = \sigma\left(\sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j)\right)$$

- all the γ_{hj}^1 for $h \in \{1, \ldots, p\}$ with p $1-$ neuron networks $\mathcal{L}^{\gamma_h^1} : \mathbb{N}^{2^{2p}+2} \longrightarrow \mathbb{N}$ which compute

$$o^{\gamma_{hj}^1} = \sigma\left(g_h + \sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j) - 1\right)$$

- all the γ_{hj}^2 for $h \in \{1, \ldots, p\}$ with p $1-$ neuron networks $\mathcal{L}^{\gamma_h^2} : \mathbb{N}^{2^{2p}+2} \longrightarrow \mathbb{N}$ which compute

$$o^{\gamma_{hj}^2} = \sigma\left(\mathtt{push0}(g_h) + \sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j) - 1\right)$$

- all the γ_{hj}^3 for $h \in \{1, \ldots, p\}$ with p $1-$ neuron networks $\mathcal{L}^{\gamma_{hj}^4} : \mathbb{N}^{2^{2p}+2} \longrightarrow \mathbb{N}$ which compute

$$o^{\gamma_{hj}^3} = \sigma\left(\mathtt{push1}(g_h) + \sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j) - 1\right)$$

- all the γ_{hj}^4 for $h \in \{1, \ldots, p\}$ with p $1-$ neuron networks $\mathcal{L}^{\gamma_{hj}^4} : \mathbb{N}^{2^{2p}+2} \longrightarrow \mathbb{N}$ which compute

$$o^{\gamma_{hj}^4} = \sigma\left(\mathtt{pop}(g_h) + \sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j) - 1\right)$$

transforming Equation (B.7) into

$$\begin{aligned} x_i^{t+1} &= \sigma(\textstyle\sum_{j=1}^{s} o^{\beta_j}) \\ &= \sigma\left(\textstyle\sum_{j=1}^{s} \sigma\left(\sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j^t)\right)\right) \\ &= \sigma\left(\textstyle\sum_{j=1}^{s} \sigma\left(\sum_{r=1}^{2^t} c_r \sigma\left(v_r^0 + v_r^1 x_1 + \cdots + v_r^p x_p + v_r^{p+1} g_1 \cdots + v_r^{2p} g_p + v_r^{2p+1}\right.\right.\right. \end{aligned}$$

and Equation (B.8) into

$$\begin{aligned} g_h^{t+1} = \sigma(\quad & \textstyle\sum_{j=1}^{s} \sigma\left(g_h + \sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j) - 1\right) + \\ & \textstyle\sum_{j=1}^{s} \sigma\left(\mathtt{push0}(g_h) + \sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j) - 1\right) + \\ & \textstyle\sum_{j=1}^{s} \sigma\left(\mathtt{push1}(g_h) + \sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j) - 1\right) + \\ & \textstyle\sum_{j=1}^{s} \sigma\left(\mathtt{pop}(g_h) + \sum_{r=1}^{2^t} c_r o_r^j(\mathbf{read}, x_j) - 1\right) \quad) \end{aligned}$$

B.4.5 Neurons needed

It is possible to calculate the total number of neurons needed for the 3-layer network translation of a p-stack machine:

- First layer L^1: s state neurons and p stack neurons: $s + p$ neurons
- Second Layer L^2: copy of the s states x_i and p stacks g_h plus p top and p non-empty: $s + 3p$
- Third Layer L^3 of $\mathcal{L}^r$ nets: $s \cdot 2^{2p}$ neurons plus p top and p non-empty: $s \cdot 2^{2p} + p$

Thus the total number of neurons composing a Rational Neural Network which implements a $p-$stack machine with s states is

$$Tot(s, p) = s + p + s + 3p + s \cdot 2^{2p} + p = 2s + s \cdot 2^{2p} + 4p$$

B.4.6 Input and Output

1. the input $I^{ext} \equiv (I^D(t), I^V(t)) \in \{0,1\}^2$ of the network is constituted of two lines:

- $I^D(t)$ is the *Data Line*, in which there is a binary encoded sequence of 0 and 1 constituting the binary input of the network
- $I^V(t)$ is the *Validation Line*, which assumes the value 1 when I^D has to be "read" from the network and 0 when I^D is deactivated

1. the output is similarly read on two neurons $O \equiv (O^D(t), O^V(t))$ where the meaning of $O^D(t)$ and $O^V(t)$ is similar to the ones in the input

To encode a string $s = a_1, \ldots a_K \in \{0,1\}^K \subseteq \{0,1\}^+$ it is possible to write

$$I^s = (I^{D_s}, I^{V_s})$$

where

$$I^{D_s}(t) = \begin{cases} a_k & \text{if } t \in \{1, \ldots, K\} \\ 0 & \text{otherwise} \end{cases}$$

and

$$I^{V_s}(t) = \begin{cases} 1 & \text{if } t \in \{1, \ldots, K\} \\ 0 & \text{otherwise} \end{cases}$$

This encoding will be denoted with $I^{D_s} = s0^\infty$, and $I^{V_s} = 1^{|s|}0^\infty$.

A partial function

$$\boldsymbol{\psi} : s \in \{0,1\}^+ \longrightarrow \{0,1\}^+$$

is computed by a net $\mathcal{N}$ if $\forall s \in \{0,1\}^+$ when

- if $\boldsymbol{\psi}(s)$ is undefined $O^{V_s} = 0^\infty$
- if $\boldsymbol{\psi}(s)$ is defined $\exists r \in \mathbb{N}$ such that the output validation line is

$$O^{V_s}(t) = \begin{cases} 1 & \text{if } t \in \{r, \ldots, r + |\boldsymbol{\psi}| - 1\} \\ 0 & \text{otherwise} \end{cases}$$

and the output data line is

$$O^{D_s}(t) = \begin{cases} \psi_{t-r+1}(s) & \text{if } t \in \{r, \ldots, r + |\boldsymbol{\psi}| - 1\} \\ 0 & \text{otherwise} \end{cases}$$

B.4.7 Turing Universalityv for $\mathbb{Q}NN$s

It is possible to reduce the number of the networks leading to the $p-$stack realization

$$Tot_2(s,p) = s + 12p + s \cdot 3^p + 2p = s + s \cdot 3^p + 14p + 2$$

In (Shannon, 1956) it was proved that there exists a $p-$stack machine whith 2 stacks and 84 states, which is able to simulate the Universal Turing Machine.

Moreover Input / output computations needs to add an extra 16 neurons, thus obtaining a total number of 886 neurons sufficient for a $\mathbb{Q}NN$ to implement a Universsal Turing Machine.

C

Analysis of Dynamical System by abstraction

Dynamical systems are a powerful instrument for modelling in robotic and biological framework. In the seminal work of Beer in the middle years of the '90s (Beer, 1995a), it was proposed to consider an entire robotic system (a robot interacting with its environment) as coupled *dynamical systems*. In this way a framework in which the entire system results in a *closed* system is defined. In each subsystem variables are strongly coupled, while interactions between systems are captured by changing the parameters which each system forces on the other. However dynamical system has become a common language also in neuroscience and in biological modelling in general (Sontag, 1990). Distributed and neuron-like phenomena are well-captured in this framework: as it is pointed out in (Izhikevich, 2004) information processing in neurons can be studied considering their *dynamical properties* and modelling each cell's bifurcation dynamics. In dynamical systems we are free to choose the desired level of details, without having to deal with coarse assumptions: as we sketched before it is possible to model high level discrete logical phenomena together with low level continuous dynamics capturing finer grain de-

tails when necessary. In other words dynamical systems allow one to describe the phenomena in study with the desired level of abstraction.

A crucial aspect of the designing process is to keep a balance between:

- *flexibility*; trying to capture each detail is the easiest way to the modelling approach. However the more details we include the more complex the system becomes;
- *rigorousness*; the modelled system has to be parametrically *controllable* and properties of interest should be verified in order to gain knowledge on the constructed system. Known limits on the system could suggest designing improvement and capturing of new phenomena.

These two aspects result in an unstable equilibrium, because the more we let the system explode in complexity, the more difficult the control and verification of important features become. The increasing model complexity has to be carefully justified when causing the lacking of controlling capabilities.

Abstraction is a powerful method which allows to deal with smaller systems simpler to be analyzed. The abstraction method can be formally applied by means of different kinds of *simulation* relations present in literature. In computer science simulation is usually based on the notion of "a machine which performs the same computation". However this would imply a state by state comparison that is not so useful when we want to abstract simpler systems. In a dynamical system the notion of simulation is captured by topological equivalence as we showed in Chapter (5): this allows the definition of different kinds of simulation relation preserving different kinds of properties in exam.

The study of complex systems needs, however, to find general simulation relations which do not turn out to be "homogeneous" in the sense that they can be generalized to compare not only continuous to continuous systems or discrete to discrete systems, but hybrid to hybrid ones.

When exact simulation properties cannot be established for the system in case, at least approximate simulation properties should be established in order that the *synthetic* abstracted system results in an ideal set of behaviours to which within a certain tolerance the *real* system tends. The distributed aspect of the system in study guides us to take into account *compositionality* and *scalability* aspects. We would like to construct systems which, when small (with respect to the total number of elements) parts of them are removed or added, would not loose some important properties gained by the initial ones. The aim is that clever composition of systems results in the possibilities of preserving old properties and acquiring new ones, that is a stability also in the process of gaining new information (*learning*). On the other side this should little affect the computational cost of analysis and synthesis in order to preserve the tractability of the system.

When the resulting abstraction has the strong property of being in direct comparison with computational systems (as in the case of *discrete* abstractions) the analysis is straightforward: simulation relations established with subclasses of computational systems result in computational properties of the starting simulated systems. Such systems could, in some mathematically quantified sense, show a finite language by which they can turn out to be *programmable* with respect to the class of systems that they are capable of simulating.

As an ulterior consequence of such a discrete description of systems, their behavioural properties could be analyzed in an *algorithmic* way. This leads to the definition of formal verification tools enabling an *exhaustive* research in the state space of a suitable model of the system to be analyzed with respect to the specification of a property φ. Different algorithms have been proposed in order to decide properties of different subclasses of dynamical systems. In fact in order to be practically applied in an automatic way, interesting subclasses have to be singled out and characterized. Even when in such a general framework the possibilities of expressing very complex systems cause many problems to be *undecidable,* the discovery of important properties, such as detection of anomalies, could at least be *guided* or some modelling restrictions could be singled out in order to apply algorithmic procedures. This means that suitable *finite abstraction* of the continuous part of the model should carefully be extracted, trying to preserve properties of the original system which is under investigation. In robotics and biology different qualitative models can be used to describe the same system at different levels of detail: the various levels can formally be related with levels of abstraction preserving certain kinds of properties of interest. In this sense formal verification techniques like model checking allow an effective manner of controlling behavioural properties. In this appendix we will show how the abstraction of a class of dynamical systems, $O-$minimal dynamical systems turns out to be finitely abstractable, letting us envisage how algorithms computing these abstractions could allow us in the end to systematically analyze the behaviour of such systems inside Formal Verification Techniques. In particular we show the pos-

sibility that the procedure $Bis\omega$ described in this appendix terminates, so that we can obtain a finite abstraction of the system in exam. However it must be stressed that even though we do know that the procedure terminates, we do not know a general algorithm which actually implements the procedure.

C.1 Analysis by abstraction

The concept of abstraction can be made rigorous (see also Section 5.2) if it can be related to the concept of simulation and then equivalence. Intuitively equivalence induces partitions and partitions induce abstraction (Zhang, 1994).

Definition C.1.1. A binary relation $\sim \subseteq A^2$ is an *equivalence relation* if it satisfies the properties

- $\forall x \in A\ (x, x) \in A$ (*riflexivity*)
- $\forall x, y \in A\ (x, y) \in \sim \iff (y, x) \in \sim$ (simmetry)
- $\forall x, y, z \in A (x, y) \in \sim \wedge (y, z) \in \sim \implies (x, z) \in \sim$ (transitivity)

Definition C.1.2. An *equivalence class* of an element $a \in A$ given by an equivalence relation $\sim$ is given by $[a] = \{x \in A\ : x \sim a\}$

An equivalence relation on A induces a partition on the quotient set $X/\sim$.

Definition C.1.3. A *language equivalence* for a transition system $T \equiv (Q, \Sigma, \rightarrow, Q_0)$ is an equivalence relation $\sim_L$ on Q such that $\forall q_1, q_2 \in Q$

- $q_1 \sim_L q_2 \implies L(q_1) = L(q_2)$

Definition C.1.4. A *partial simulation* between two transition systems $T \equiv (Q, \Sigma, \rightarrow, Q_0)$ and $T^{'} \equiv (Q^{'}, \Sigma^{'} \equiv \Sigma, \rightarrow', Q_0^{'})$ is an equivalence relation $\sim \subseteq Q \times Q^{'}$ such that

$\forall q_1, q_2 \in Q$, $\forall q_1' \in Q'$,$\forall a \in \Sigma$
$(q_1 \sim q_1^{'}$ and $q_1 \xrightarrow{a} q_2) \implies (\exists q_2^{'} \mid q_2 \sim q_2^{'}$ and $q_1^{'} \xrightarrow{a}{}' q_2^{'})$
We will say that T partially simulates $T^{'}$

Definition C.1.5. A *simulation* between two transition systems $T \equiv (Q, \Sigma, \phi, Q_0)$ and $T^{'} \equiv (Q^{'}, \Sigma^{'} \equiv \Sigma, \phi^{'}, Q_0^{'})$ is an equivalence relation $\sim \subseteq Q \times Q^{'}$ such that T partially simulates $T^{'}$and $\forall q_1 \in Q$ there exists p_1 such that $q_1 \sim p_1$

We will say that T simulates $T^{'}$

Definition C.1.6. A *bisimulation* between two transition systems $T \equiv (Q, \Sigma, \rightarrow, Q_0)$ and $T^{'} \equiv (Q^{'}, \Sigma^{'} \equiv \Sigma, \rightarrow', Q_0^{'})$ is an equivalence relation $\sim \subseteq Q \times Q^{'}$ such that,

- $\forall q_1, q_2 \in Q$, $\forall q_1' \in Q' \forall a \in \Sigma (q_1 \sim q_1^{'}$ and $q_1 \xrightarrow{a} q_2) \implies (\exists q_2^{'} \mid q_2 \sim q_2^{'}$ and $q_1^{'} \xrightarrow{a} q_2^{'})$
- $\forall q_1', q_2' \in Q$', $\forall q_1 \in Q \forall a \in \Sigma$ $(q_1 \sim q_1^{'}$ and $q_1^{'} \xrightarrow{a} q_2^{'}) \implies (\exists q_2 \mid q_2 \sim q_2^{'}$ and $q_1 \xrightarrow{a} q_2)$

that is T_1 simulates T_2 and T_2 simulates T_1.

We will say that T_1 and T_2 are *bisimilar.*

Corollary C.1.7. *A* bisimulation $\sim$ *for a transition transition system* $T \equiv (Q, \Sigma, \rightarrow, Q_0)$ *is an equivalence relation* $\sim \subseteq Q \times Q$ *such that* $\forall q_1, q_1^{'} q_2 \in Q$

- $\forall a \in \Sigma$ $(q_1 \sim q_1^{'}$ *and* $q_1 \xrightarrow{a} q_2) \implies (\exists q_2^{'} | q_2 \sim q_2^{'}$ *and* $q_1^{'} \xrightarrow{a} q_2^{'})$

Definition C.1.8. A *bisimulation* $\sim$ for a transition transition system $T \equiv (Q, \Sigma, \rightarrow, Q_0)$ respects a partition $\mathcal{P}$ of Q if

- $\forall q_1, q_2 \in Q \ (q_1 \sim q_2) \implies (\exists P \in \mathcal{P} \text{ with } q_1, q_2 \in P)$

Definition C.1.9. The *identity bisimulation* for a transition systems T is

- $\sim = \{(q, q) \mid q \in Q\}$

Theorem C.1.10. *Given a transition system* $T = (Q, \Sigma, \rightarrow, Q_0)$*, and a partition of* $\mathcal{P}$ *of* Q *and a bisimulation* β *with respect to* $\mathcal{P}$*, it is always possible to construct from* β *a bisimulation* $\tilde{\beta}$ *which is an equivalence relation*

C.1.1 Logical preliminaries

We put beforehand logical preliminaries that are needed to define a class of dynamical properties with the stunning property of being finitely abstractable: $O-$minimal dynamical systems.

Definition C.1.11. Non-logical symbols:

- constant symbols
- $n-$ary relations (e.g. $\leq$)
- $n-$ary operations (e.g. $+, *$)

Definition C.1.12. Logical symbols (set *conn*):

- propositional connectives ($\neg$,$\vee$, etc)
- quantifiers ($\forall$, $\exists$)
- equality ($=$)
- variables (x)

Definition C.1.13. A *structure* is an ordered pair $\mathcal{A} \equiv (|A|, I)$ where

- $|A|$ is a nonempty set, called the *universe* of A;
  - I is an *interpretation* function $I \in |A|$ whose domain is a set of non-logical symbols.
    - The domain of I is called the signature of A, $sig(A)$.
    - To each $n-$ary relation symbol $R \in sig(A)$ we assume that I assigns an $n-$ary relation $R \subseteq |A|^n$
    - To each $n-$ary operation symbol $R \in sig(A)$ we assume that I assigns an $n-$ary operation $o : |A|^n \longrightarrow |A|$
    - To each constant symbol $c \in sig(A)$ we assume that I assigns an individual constant $c \in |A|$

Definition C.1.14. A *term* t of a structure $\mathcal{A}$ is defined recursively:

- *Every constant* c *and every variable* v *is a term*
- $\forall t_1, \ldots, t_n$ terms, and $n-$ary operation symbol $o(t_1, \ldots, t_n)$ is a term

Definition C.1.15. An *atomic formula* a of a structure $\mathcal{A}$ is defined recursively:

- $\forall t_1, t_2$ terms, $t_1 = t_2$ is an atomic formula
- $\forall t_1, \ldots, t_n$ terms, and $n-$ary relation $R(t_1, \ldots, t_n)$ is an atomic formula

Definition C.1.16. The set of *formulas* $F(\mathcal{A})$ *of* a structure $\mathcal{A}$ is the minimum set satisfying:

- $\forall a$ atomic formula, $a \in F(\mathcal{A})$

- $\perp \in F(\mathcal{A})$
- if $\phi \in F(\mathcal{A})$, $\neg\phi \in F(\mathcal{A})$, $\exists x\phi$ (with x a variable)
- if $\phi_1, \phi_2 \in F(\mathcal{A})$, $\phi_1 \vee \phi_2 \in F(\mathcal{A})$

Definition C.1.17. A *sentence* is a formula with no free variables.

Definition C.1.18. A formula ϕ with free variables $\bar{x} = (x_1, \ldots, x_k)$ satisfies $\mathcal{M}$, $\mathcal{M} \models \phi(\bar{a})$, if given $\bar{a} = (a_1, \ldots, a_k) \in \mathcal{M}^k$ we obtain:

- if $\phi = t_1 = t_2$ then $\mathcal{M} \models \phi$ if $t_1^{\mathcal{M}}(\bar{a}) = t_2^{\mathcal{M}}(\bar{a})$ where $I(t, \bar{a}) = t^{\mathcal{M}}(\bar{a})$ is the result of the intepretation of the term
- if $\phi = R(t_1, \ldots, t_k)$ then $\mathcal{M} \models \phi(\bar{a})$ if $(t_1^{\mathcal{M}}(\bar{a}), \ldots, t_k^{\mathcal{M}}(\bar{a})) \in R^{\mathcal{M}}$
- if $\phi = \neg\psi$ then $\mathcal{M} \models \phi(\bar{a})$ if $\mathcal{M} \not\models \phi(\bar{a})$
- if $\phi = \psi_1 \vee \psi_2$ then $\mathcal{M} \models \phi(\bar{a})$ if $\mathcal{M} \models \phi_1(\bar{a})$ or $\mathcal{M} \models \psi_2(\bar{a})$
- if $\phi = \exists x\psi(x)$ then $\mathcal{M} \models \phi(\bar{a})$ if there exists $b \in \mathcal{M}$ such that $\mathcal{M} \models \psi(\bar{a}, b)$

Definition C.1.19. A *Theory* $Th(\mathcal{A})$ on a structure $\mathcal{A}$ is a set of sentences produced by formulas on $\mathcal{A}$.

Definition C.1.20. A *Model* $\mathcal{M}$ of a Theory $T = Th(\mathcal{A})$, $\mathcal{M} \models T$ if for all sentences ϕ in T we have $\mathcal{M} \models \phi$.

Definition C.1.21. Two structures $\mathcal{M}$ and $\mathcal{N}$ are elementarily equivalent $\mathcal{M} \equiv \mathcal{N}$ if and only if

$$Th(\mathcal{M}) = Th(\mathcal{N})$$

Theorem C.1.22. *$Th(\mathcal{M})$ is invariant under isomorphism (see Marker, 2002)*

C.2 $O-$minimal systems

In this section a picture of $O-$minimal systems is sketched. An exhaustive introduction to $O-$minimal systems can be found (Van Den Dries, 1998).

Definition C.2.1. A set A is *definable* on M if there exists an $n-$ary relation $R \subseteq M^n$ such that

$$A = \{a_1, \ldots, a_n \mid M \vDash R(a_1, \ldots a_n)\}$$

Example C.2.2. Let's consider the system

$$\begin{cases} a_{11}x_1 + a_{12}x_2 > c_1 \\ a_{21}x_1 + a_{22}x_2 \leq c_2 \end{cases}$$

It is possible to construct the formula

$$\phi\left(x_1, x_2, a_{11}, a_{12}, a_{21}, a_{22}, c_1, c_2\right) = \exists x_1 \exists x_2 \left(\left(a_{11}x_1 + a_{12}x_2 > c_1\right) \wedge \left(a_{21}x_1 + a_{22}x_2 \leq \right.\right.$$

such that we obtain the definable set $A \subseteq \mathbb{R}^6$ on $\mathbb{R}^8$

$$A \equiv \{(a_{11}, a_{12}, a_{21}, a_{22}, c_1, c_2) \,|\mathbb{R}^8 \models \phi\left(x_1, x_2, a_{11}, a_{12}, a_{21}, a_{22}, c_1, c_2\right)\}$$

Example C.2.3. Consider the definable set

$$S \equiv \{x \in \mathbb{R} \mid \sin\left(\pi x\right) = 0\}$$

Note that $S = \mathbb{Z}$

Definition C.2.4. A function $f : A \in \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is definable if its graph $\Gamma \in \mathbb{R}^{n+m}$ is definable.

Definition C.2.5. Cells are defined inductively as follows

- the cells in $\mathbb{R}$ are just the points $\{r\}$ and the intervals (a, b)
- if $C \subseteq \mathbb{R}^n$ is a cell, if $f, gC \longrightarrow \mathbb{R}^n$ are definable continuous functions and $f < g$ on C, then

$$(f, g) \equiv \{(x, r) \in C \times \mathbb{R} \, : \, f(x) < r < g(x)\}$$

  is a cell in $\mathbb{R}^{n+1}$; also the stes $\Gamma(f)$, $(-\infty, f) \equiv \{(x, r) \in C \times \mathbb{R} : r < f(x)\}$, $(f, +\infty) \equiv \{(x, r) \in C \times \mathbb{R} : f(x) < r\}$ are cells in $\mathbb{R}^{n+1}$; $C \times \mathbb{R}^{n+1}$ is a cell

Definition C.2.6. Cell decomposition theorem. Each definable set $A \subseteq \mathbb{R}^n$ has a finite partition $A = C_1 \cup C_2 \cup \cdots \cup C_k$ into cells C_i. If $f : A \longrightarrow \mathbb{R}^n$ is a definable map, this partition of A can moreover be chosen such that all restriction $f|C_i$ are continuous.

Definition C.2.7. A totally ordered structure $\mathcal{M} \equiv (M, <, \dots)$ is *o-minimal* if every definable subset of M is a finite union of points and open intervals (possibly unbounded)

Definition C.2.8. An o-minimal dynamical system is a pair $D \equiv (M, \gamma)$ where

- $M = (|M|, +, 0, 1, \dots)$ is an o-minimal totally ordered structure

  $-\ \gamma : V_1 \times M^+ \longrightarrow V_2$ is a definable function in M

  where $M^+ = \{m \in M \mid m \geq 0\}, V_1 \subseteq M^{k_1}, V_2 \subseteq M^{k_2}$

The following tables compares different $o-$minimal theories together with the trajectories and sets which they subsume.

Theory	Definable Sets	Definable Trajectories
$(\mathbb{R}, <, +, -)$	Polyhedral (Semilinear) sets	Linear trajectories
$(\mathbb{R}, <, +, -, \cdot)$	Semialgebraic sets	Polynomial trajectories
$(\mathbb{R}, <, +, -, \cdot, \{\hat{f}\})$	Subanalytic sets	Polynomial trajectories
$(\mathbb{R}, <, +, -, \cdot, exp)$	Semialgebraic sets	Exponential trajectories
$(\mathbb{R}, <, +, -, \cdot, exp, \{\hat{f}\})$	Subanalytic sets	Exponential trajectories

C.3 Control in o-minimal dynamical systems

Definition C.3.1. An o-minimal hybrid game is a tuple $\mathcal{G} = (Q, Goal, \Sigma, \delta, \gamma, G, R)$ on an o-minimal structure $\mathcal{M} = (M, +, 0, 1, \dots)$ where

- $\mathcal{H} = (Q, \Sigma, \delta, \gamma, G, R)$ is an o-minimal hybrid system
- $Goal \subseteq Q$ is a subset of *winning* locations
- Σ is partitioned in Σ_c and Σ_u corresponding to *controllable* and *uncontrollable* actions.

Remark C.3.2. The game is played between two players, the *controller* and the *environment*; the goal of the controller is to reach a winning state whatever the environment does. In every state s, the controller picks a delay τ and an action $c \in \Sigma_c$ so as to hope in a transition $s \xrightarrow{\tau,c} s'$. The environment has two choices

- either it waits and executes the transition $s \xrightarrow{\tau,a} s'$ proposed by the controller
- or it waits τ', $0 \leq \tau' \leq \tau$, and executes a transition $s \xrightarrow{\tau,u} s'$

The game evolves in a new state according to the choice of the environment.

Definition C.3.3. An action $(\tau, a) \in M^+ \times \Sigma$ is *enabled* in a state (q, x, t, y) if there exist (q', x', t', y') and (q'', x'', t'', y'') such that $(q, x, t, y) \xrightarrow{\tau} (q', x', t', y') \xrightarrow{a} (q'', x'', t'', y'')$. We write $(q, x, t, y) \xrightarrow{\tau, a} (q'', x'', t'', y'')$. The set of all action enabled in a state (q, x, t, y) is $Enb(q, x, t, y)$

Definition C.3.4. A *run* ρ in $\mathcal{H}$ is a (finite or infinite) sequence

$$\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} (q_1, x_1, t_1, y_1) \xrightarrow{\tau_2, a_2} \dots$$

Definition C.3.5. A *position* along ρ is a pair $(i, \tau) \in \mathbb{N} \times M^+$ such that $\tau \leq \tau_{i+1}$

- $\rho_i = (q_i, x_i, t_i, y_i)$
- $\rho[(i, \tau)] = (q_i, \gamma_{q_i}(x_i, t_i + \tau))$
- $\rho_{\leq (i,\tau)} = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \dots \xrightarrow{\tau_{i-1}, a_{i-1}} (q_i, x_i, t_i, y_i) \xrightarrow{\tau} (q_i, x_i, t_i, \gamma_{q_i}(x_i, t_i + \tau))$
- $\rho_{\geq (i,0)} = (q_i, x_i, t_i, y_i) \xrightarrow{\tau_{i+1}, a_{i+1}}$
- if ρ is a finite run of length n, $last(\rho) = (q_n, x_n, t_n, y_n)$

Definition C.3.6. $Runs_f(\mathcal{G})$ is the set of all finite runs of $\mathcal{G}$

Definition C.3.7. $Runs(\mathcal{G})$ is the set of all finite and infinite runs of $\mathcal{G}$

Definition C.3.8. A *controller* (or a *strategy*) λ is a partial function

$$\lambda : Runs_f(\mathcal{G}) \longrightarrow M^+ \times \Sigma_c$$

such that for all $\rho \in Runs_f(\mathcal{G})$, if $\lambda(\rho)$ is defined then $\lambda(\rho)$ is enabled in $last(\rho)$

Definition C.3.9. A run $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \dots$ is *consistent with a strategy* λ, when $\forall i$, if $\lambda(\rho_i) = (\tau, a)$ then either $\tau_{i+1} = \tau$ and $a_{i+1} = a$ or $\tau_{i+1} \leq \tau$ and $a_{i+1} \in \Sigma_u$. $Outcome(s, \lambda)$ is the set of all the runs starting from the state s and consistent with the strategy λ.

Definition C.3.10. A run $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \dots$ is *winning* if for some i,$q_i \in Goal$

Definition C.3.11. A run ρ is said to be *maximal* with respect to a strategy λ if it is either infinite, or $\lambda(\rho)$ is not defined.

Definition C.3.12. A strategy λ is *winning from a state* (q, y) if $\forall (x, t)$ such that $\gamma_q(x, t) = y$ all maximal runs consistent with λ are winning.

Problem C.3.13. Control Problem in o-minimal games. Given an o-minimal game $\mathcal{G}$ and a definable initial state (q, y), determine if there exists a winning strategy λ starting from (q, y)

Definition C.3.14. An event $(q, x, t, , y)$ can be reached from (q_0, x_0, t_0, y_0) iff there exists a path $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \dots \xrightarrow{\tau_n, a_n} (q, x, t, y)$. Or equivalently (q_0, x_0, t_0, y_0) can be controlled to $(q, x, t, , y)$

Definition C.3.15. A state (q, y) can be reached from (q_0, y_0) iff there exists a path $\rho = (q_0, x_0, t_0, y_0) \xrightarrow{\tau_1, a_1} \dots \xrightarrow{\tau_n, a_n} (q, x, t, y)$. Or equivalently (q_0, y_0) can be controlled to (q, y)

C.4 Bisimulation algorithm

The bisimulation algorithm is a general procedure in order to find a bisimulation. (see (C.1)). In this section we will refor-

mulate it for dynamical systems through the computation of suffix dynamical types (Brihaye, 2006).

Definition C.4.1. Predecessor Set

$$Pre_t(P) = \{y_0 \in V_2 \,|\, \exists y \in P \ \wedge y_0 \xrightarrow{t} y\}$$

Algorithm C.1 Bisimulation Algorithm

`inizialization:` $\tilde{Q} := \mathcal{P}$
`while` $\exists t \in \Sigma \ \exists P, P' \in \tilde{Q}$ `such that` $\emptyset \neq P \cap Pre_t(P') \neq P$
    `set` $P_1 = P \cap Pre_t(P)$ `and` $P_2 = P \backslash Pre_t(P)$
    `refine` $\tilde{Q} := \tilde{Q} \backslash \{P\} \cup \{P_1, P_2\}$

Definition C.4.2. $\mathcal{F}_x = \{I \,|\, I$ is an interval maximal for $\exists P \in \mathcal{P}, \forall t \in I, \gamma(x,t) \in P\}$

Definition C.4.3. I_t is the interval in $\mathcal{F}_x$ such that $\gamma(x,t) \in I_t$

Definition C.4.4. A word ω_{x_0} on a partition $\mathcal{P}$ of the set Y from a dynamical system $D = (X, \gamma, T)$ is the succession of sets of the partition $\mathcal{P}$

$$\omega_{\mathbf{x}_0} \ : \ \mathcal{F}_{\mathbf{x}_0} \longrightarrow \mathcal{P}$$

where $\mathcal{F}_{\mathbf{x}_0}$ is a succession of intervals or points of T, determined by the trajectory $\gamma(t, \mathbf{x}_0)$, of the induced partition $\mathcal{F}$ on T constructed as $\{t \in T \,|\, \gamma(t, \mathbf{x}_0) \in \mathcal{P}\}$.

Definition C.4.5. We denote by $\Omega_{\mathcal{P}}$ the set of words associated with the dynamical system (X, γ, T) with respect to a partition $\mathcal{P}$

The set $\Omega_{\mathcal{P}}$ gives a complete static description of the dynamical system

Definition C.4.6. Given the set of intervals

$$\mathcal{F}_{(\mathbf{x},t)} = \{I \in \mathcal{F}_{\mathbf{x}} \mid I \geq I_t\}$$

the *suffix* of the world $\omega_{\mathbf{x}}$ associated with time t is the restriction

$$\omega_{(\mathbf{x},t)} = \omega_{\mathbf{x}}|_{\mathcal{F}_{(\mathbf{x},t)}}$$

Definition C.4.7. The *suffix dynamical type* of $y \in Y$ with respect to a partition $\mathcal{P}$ of V_2, given a dynamical system (X, γ, T), is defined by

$$\mathrm{Suf}_{\mathcal{P}}(y) = \{\omega_{(\mathbf{x},t)} \mid \gamma(t, \mathbf{x}) = y\}$$

Definition C.4.8. The *suffix partition* with respect to a partition $\mathcal{P}$ of a dynamical system (D, γ, T) is the partition induced by the equivalence relation on the phase space Y between points *having the same suffix dynamical type.*

The Bisimulation algorithm ensures that either $\mathcal{P} \equiv Suf(\mathcal{P})$ or $Suf(\mathcal{P})$ refines $\mathcal{P}$. In the former case we obtain the bisimulation $\mathcal{P}$. In the latter we iterate the algorithm until $Suf^i(\mathcal{P})$. This let us write a new version of the bisimulation algorithm, procedure $Bis\omega$ (see Algorithm (C.2))

Lemma C.4.9. *(Brihaye, 2006) Given a dynamical system D and a partition $\mathcal{P}$ of its phase space iterating the partition induced by Suf we obtain*

$$\mathcal{P} \prec Suf(\mathcal{P}) \prec Suf^2(\mathcal{P}) \prec \cdots \prec Suf^k(\mathcal{P}) \prec \ldots$$

Algorithm C.2 Suffix Procedure

inizialization: $\mathcal{P} := \mathcal{P}_0$
    continue:=TRUE
do
    compute the set of the words $\Omega_{\mathcal{P}}$
    associate $Suf_{\mathcal{P}}(y)$ with each $y \in V_2$ and Build $Suf(\mathcal{P})$
    if $\mathcal{P} = Suf(\mathcal{P})$
        then continue:=FALSE;
        else $\mathcal{P} := Suf(\mathcal{P})$
while (continue)
return $\mathcal{P}$

Example C.4.10. Consider the system

$$\frac{dx}{dt} = -x$$

Calculating the transition system

$$\int_{x_0}^{x(t)} \frac{dx}{x} = . \int_{t_0}^{t} dt$$

then

$$\ln(x(t)) - \ln(x_0) = -(t - t_0)$$

from which

$$x(t) = x_0 e^{t_0} e^{-t}$$

$$\gamma_{x_0}(x, t) = x_0 e^{t_0} e^{-t}$$

Given the partition $\mathcal{P} = \{A = \{x \in \mathbb{R} \,:\, x < 1\}, B = \{x \in \mathbb{R} \,:\, x \geq 1\}\}$

$Suf_{\mathcal{P}}(y \in A) = \{A\}$, $Suf_{\mathcal{P}}(y \in B) = \{B, A\}$

$$Suf(\mathcal{P}) = \{A, B\}$$

C.5 O-minimal system and Feed Forward Networks

Using arguments from model theory in (Sontag, 1996) it is proved that for a functional quadratic loss between a dataset of N labelled examples and the output of different types of Neural Networks find an upper bound for the number of critical points, local minima of the functional, assuming that $N \geq 2K(m+2)+3$ (i.e. enough samples to make the problem not undetermined). Now we use the same arguments from Model Theory to study the possibility of applying $Bis\omega$ algorithm to neural networks.

C.6 Bisimulation procedure applied

We will find bisimulation for increasing complex systems in order to understand how the *Bisimulation procedure* could be applied.

C.6.1 Single 'feed forward' Neuron

Let us firstly consider a simple network (see Figure C.1) of only one neuron with a single input. So let us consider the equation:

$$\gamma_{single}(x, w, \theta) = w \cdot \sigma(x + \theta) \tag{C.1}$$

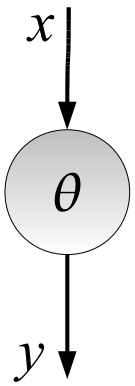

Figure C.1: Single 'feed forward' neuron

where $\sigma = 1/(1+e^{-x})$. If we consider the order-minimal theory $(\mathbb{R}, \leq, +, \cdot, e^x, 0, 1)$ we are able to build an o-minimal dynamical system:

Lemma C.6.1. *Let* $R = (\mathbb{R}, \leq, +, \cdot, e^x, 0, 1)$ *and* $\gamma_{single}(x, w, \theta) = w \cdot \sigma(x+\theta)$. *Then* (R, γ_{single}) *is an o-minimal dynamical system*

Proof. The graph of γ is the set

$$G = \{(x, w, \theta, \gamma(x, w, \theta)\}$$

As we can write

$$G = \{(x, w, \theta, y) \in \mathbb{R}^4 \, : \, y \cdot (e^{x+\theta} + 1) = w \cdot e^{x+\theta}\}$$

in fact we can write

$$y = w \cdot \sigma(x + \theta) = w \cdot \frac{1}{1 + e^{-x-\theta}}$$

and then

$$y \cdot (1 + e^{-x-\theta}) = w$$

from which

$$y \cdot \frac{e^{+x+\theta}}{e^{+x+\theta}}(1 + e^{-x-\theta}) = y \cdot \frac{(e^{+x+\theta} + 1)}{e^{+x+\theta}} = w$$

and finally

$$y \cdot (e^{x+\theta} + 1) = w \cdot e^{x+\theta}$$

Thus

$$\phi(x, w, \theta, y) = y \cdot (e^{x+\theta} + 1) = w \cdot e^{x+\theta}$$

is clearly definable and is clearly a formula in the *o*-minimal theory $(\mathbb{R}, \leq, +, \cdot, e^x, 0, 1)$. □

C.6.2 Bisimulation for a single neuron

Let us consider the simplified equation $\gamma : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{R}$

$$\gamma(x, t) = (x, \gamma_{single}(x, t))$$

with $\gamma_{single}(x, t) = x \cdot \sigma(t)$. Let us consider the order-minimal theory $R = (\mathbb{R}, \leq, +, \cdot, e^x, 0, 1)$. The system (R, γ) is clearly an o-minimal dynamical system.

Then, to start with, consider the partition (see Figure C.2) of $\mathbb{R}^2$, $\mathcal{P}^0 = \{P_1^0, P_2^0\}$ with

- $P_1^0 = \{(-\infty, \infty) \times [2, 3]\}$
- $P_2^0 = \{\mathbb{R}^2 \backslash P_1^0\} = \{(-\infty, \infty) \times (-\infty, 2) \cup (-\infty, \infty) \times (3, \infty)\}$

Let us apply by hand the procedure of suffix partition.

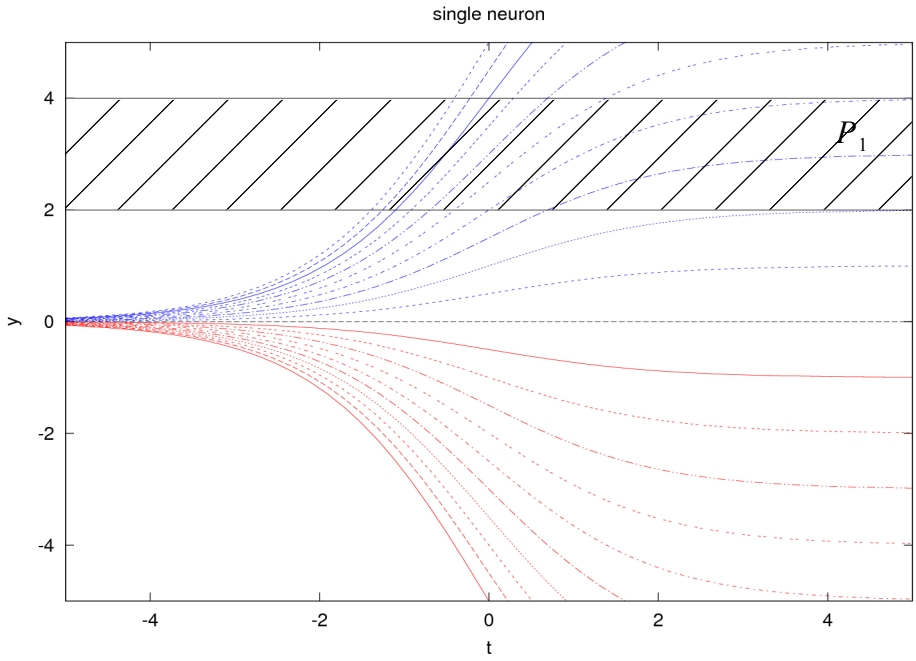

Figure C.2: Single Neuron trajectories and partition space.

Suffix partition for a single neuron

Let us apply the $Bis\omega$ procedure. Having the partition $\mathcal{P}^0 = \{P_1^0, P_2^0\}$, let us assign the simbol a to states in partition P_1^0 and symbol b to states in partition P_2^0.

- $\mathcal{F}_x = \{I \mid I \text{ is an interval maximal for } \exists P \in \mathcal{P}^0, \forall t \in I, \gamma_{single}(x,t) \in P\}$
  - e.g. $\mathcal{F}_{x_0=5} = \{(-\infty, \sigma^{-1}(2/5)), [\sigma^{-1}(2/5), \sigma^{-1}(4/5)], (\sigma^{-1}(4/5), \infty)\}$
- I_t is the interval in $\mathcal{F}_x$ such that $\gamma_{single}(x,t) \in I_t$
  - e.g. in $\mathcal{F}_{x_0=5}$ we have $I_{t=0.5} = [\sigma^{-1}(2/5), \sigma^{-1}(4/5)]$
- $\mathcal{F}_{(x,t)} = \{I \mid I \in \mathcal{F}_x \wedge I \geq I_t\}$
  - e.g. $\mathcal{F}_{(x_0=5,t=0.5)} = \{[\sigma^{-1}(2/5), \sigma^{-1}(4/5)], (\sigma^{-1}(4/5), \infty)\}$
- associating to each $P_i \in \mathcal{P}$ a symbol a_i we obtain $\omega_x(I) = a_i$ with $I \in \mathcal{F}_x$
  - e.g. associating a with P_1^0 and b with P_2^0, $\omega_{x_0=5}((-\infty, \sigma^{-1}(2/5)) = b$
- Given $\mathcal{F}_x = \{I_1, \ldots, I_n\}$, with $n \in \mathbb{N}$, $\omega_x = b_1 \ldots b_n$ with $b_i = \omega_x(I_i)$
  - e.g. $\omega_{x_0=5} = bab$
- the *suffix* of the world ω_x associated wih time t is the restriction $\omega_{(x,t)} = \omega_x|_{\mathcal{F}_{(x,t)}}$
  - e.g. $\omega_{(x_0=5,t=0.5)} = ba$
- $Suf_{\mathcal{P}^0}(y) = \{\omega_{(x,t)} \mid \gamma_{single}(x,t) = y\}$ induces an equivalent relation of points of the output space having the same suffix dynamical type:

It is possible to see that the space can be partition into six zones

- $P_1^1 = (-\infty, \infty) \times (-\infty, 0] \cup \cup (-\infty, \infty) \times \{(t, y) \,|\, y \leq 2 \cdot \sigma(t) \wedge y \in (0, 2)\} \cup \cup (-\infty, \infty) \times (4, \infty)$
- $P_2^1 = (-\infty, \infty) \times \{(t, y) \,|\, y > 2 \cdot \sigma(t) \wedge y \leq 4 \cdot \sigma(t) \wedge y \in (0, 2)\}$
- $P_3^1 = (-\infty, \infty) \times \{(t, y) \,|\, y > 4 \cdot \sigma(t) \wedge y \in (0, 2)\}$
- $P_4^1 = (-\infty, \infty) \times \{(t, y) \,|\, y \leq 4 \cdot \sigma(t) \wedge y \in [2, 4]\}$
- $P_5^1 = (-\infty, \infty) \times \{(t, y) \,|\, y > 4 \cdot \sigma(t) \wedge y \in [2, 4]\}$

In fact it happens that

- $Suf_{\mathcal{P}^0}(P_1^1) = b$
- $Suf_{\mathcal{P}^0}(P_2^1) = ba$
- $Suf_{\mathcal{P}^0}(P_3^1) = bab$
- $Suf_{\mathcal{P}^0}(P_4^1) = a$
- $Suf_{\mathcal{P}^0}(P_5^1) = ab$

The system satisfies the hypothesis of suffix determinism, i.e. only one word is associated with each point of the phase space (see Brihaye, 2006). We can go on denoting $\mathcal{P}^1 = Suf(\mathcal{P}^0) = \{P_1^1, P_2^1, P_3^1, P_4^1, P_5^1\}$, assigning to each partition a symbol $a_i \longrightarrow P_i^1$.

It happens that

- $Suf_{\mathcal{P}^1}(P_1^1) = a_1$
- $Suf_{\mathcal{P}^1}(P_2^1) = a_2 a_4$
- $Suf_{\mathcal{P}^1}(P_3^1) = a_3 a_5 a_1$

- $Suf_{\mathcal{P}^1}(P_4^1) = a_4$
- $Suf_{\mathcal{P}^1}(P_5^1) = a_5 a_1$

so we find $Suf(\mathcal{P}^1) = \mathcal{P}^1$ or equivalently, as we expected from the property of suffix determinism, $Suf^2(\mathcal{P}^1) = Suf(\mathcal{P}^1)$ and the procedure stops.

C.6.3 Output of a single neuron

Let us consider the equation $\gamma : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{R}$

$$\gamma(x,t) = (x, \gamma_{out}(x,t))$$

with $\gamma_{out}(x,t) = \sigma(x_1 \cdot \sigma(t))$. Let us consider the order-minimal theory $R = (\mathbb{R}, \leq, +, \cdot, e^x, 0, 1)$. The system (R, γ) is clearly an o-minimal dynamical system. Note that this can be considered as the "output" of the one-neuron equation previously considered.

Then, to start with, consider the partition (see Figure C.3) of $\mathbb{R}^2$, $\mathcal{P}^0 = \{P_1^0, P_2^0\}$ with

- $P_1^0 = (-\infty, \infty) \times [0.7, 0.9]$
- $P_2^0 = \mathbb{R}^2 \backslash P_1^0 = (-\infty, \infty) \times (-\infty, 0.7) \cup (-\infty, \infty) \times (0.9, \infty)$

Reasoning in the same manner as for the single neuron case, we obtain

- $$\begin{aligned} P_1^1 &= (-\infty, \infty) \times (-\infty, 0] \cup \\ &\cup \ (-\infty, \infty) \times \{(t,y) \,|\, y \leq \sigma(\sigma^{-1}(0.7) \cdot \sigma(t)) \wedge y \in (0,2)\} \cup \\ &\cup \ (-\infty, \infty) \times (4, \infty) \end{aligned}$$

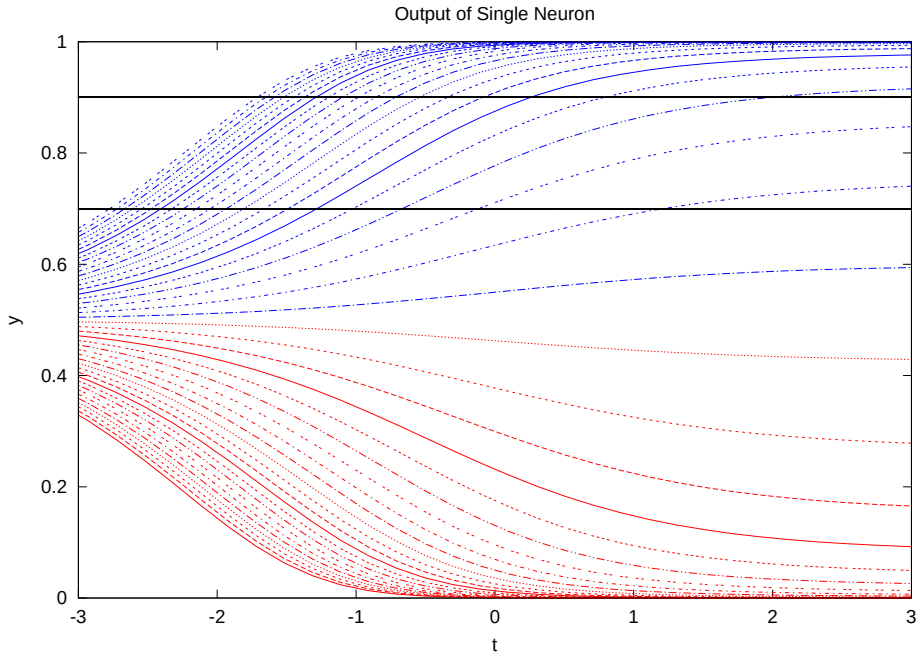

Figure C.3: Output of a single Neuron trajectories and partition space

- $P_2^1 = (-\infty, \infty) \times \{(t, y) \,|\, y > \sigma(\sigma^{-1}(0.7) \cdot \sigma(t)) \wedge y \leq \sigma(\sigma^{-1}(0.9) \cdot \sigma(t)) \wedge y \in (0, 2)\}$
- $P_3^1 = (-\infty, \infty) \times \{(t, y) \,|\, y > \sigma(\sigma^{-1}(0.9) \cdot \sigma(t)) \wedge y \in (0, 2)\}$
- $P_4^1 = (-\infty, \infty) \times \{(t, y) \,|\, y \leq \sigma(\sigma^{-1}(0.9) \cdot \sigma(t)) \wedge y \in [2, 4]\}$
- $P_5^1 = (-\infty, \infty) \times \{(t, y) \,|\, y > \sigma(\sigma^{-1}(0.9) \cdot \sigma(t)) \wedge y \in [2, 4]\}$

As we can see the

C.6.4 Network with monotonic condition

Given a network model

$$x_{out}(x_{in}) = \gamma(w_{out}^h, \ldots, w_H^h, w_1^{in}, \ldots, w_H^{in}, x_{in}) = \sum_{h=1}^{H} w_{out}^h \cdot \sigma(x_h) = \sum_{h=1}^{H} w_{out}^h \cdot \sigma(\underbrace{w_h^{in} \cdot \sigma(x_{in})}_{\theta_1}) \quad (C.2)$$

$$\frac{\partial x_{out}}{\partial x_{in}} = \sum_{h=1}^{H} w_{out}^h \cdot \frac{\partial \sigma(\theta_1)}{\partial x_{in}} = \sum_{h=1}^{H} w_{out}^h \cdot \frac{\partial \sigma(\theta_1)}{\partial \theta_1} \cdot \frac{\partial \theta_1}{\partial x_i} = \sum_{h=1}^{H} w_{out}^h \cdot \frac{\partial \sigma(\theta_1)}{\partial \theta_1} \cdot w_h^{in}$$

To ensure monotonicity of $x_{out}$with respect to x_{in}, the function has to respect

$$\frac{\partial x_{out}}{\partial x_{in}} \geq 0$$

and as $\frac{\partial\sigma(\theta_1)}{\partial\theta_1} = \sigma(\theta_1)\cdot(1-\sigma(\theta_1)) \geq 0$, the only condition to satisfy is

$$\sum_{h=1}^{H} w_{out}^{h}\cdot w_{h}^{in} \geq 0 \tag{C.3}$$

A simpler suffcient condition to obtain the same thing is to have

$$w_{out}^{h}, w_{h}^{in} \geq 0 \,\forall h \in \{1,\ldots,H\}$$

The condition (C.4) can be generalized to multiple layers where we have (Lang, 2005)

$$\sum_{l_H=1}^{L_H} w_{out}^{l_H}\cdots\sum_{l_2=1}^{L_2} w_{l_3}^{l_2}\sum_{l_1=1}^{L_1} w_{l_2}^{l_1}\cdot w_{h}^{in} \geq 0 \tag{C.4}$$

This condition can be considered in order to look for a general procedure to apply $Bis\omega$ to Networks. The monotonicity, in fact, allows to refine partitions checking the edges of the intervals.

C.6.5 Single CTRNN neuron

Considering the equation for a single CTRNN neuron:

$$\dot{y} = -y + w\cdot\sigma(y+\theta) + I$$

We do not know if the solutions of this equations are all o-minimal. In general we do not have a way to distinguish the o-minimal solution. However if we consider the DTRNN approximation:

$$y_{k+1} = y_k + \Delta t(-y_k + w\cdot\sigma(y_k+\theta) + I)$$

we can write

$$f(y; w, \theta, I) = -y + w \cdot \sigma(y + \theta) + I$$

$$y_1 = y_0 + \Delta t \cdot f(y_0)$$

then

$$\begin{aligned} y_2 &= y_1 + \Delta t \cdot f(y_1) = \\ &= y_0 + \Delta t \cdot f(y_0) + \Delta t f(y_0 + \Delta t \cdot f(y_0)) \end{aligned}$$

and

$$\begin{aligned} y_3 &= y_2 + \Delta t \cdot f(y_2) = \\ &= y_0 + \Delta t \cdot f(y_0) + \Delta t f(y_0 + \Delta t \cdot f(y_0)) + \\ &+ \Delta t \cdot f(y_0 + \Delta t \cdot f(y_0) + \Delta t f(y_0 + \Delta t \cdot f(y_0))) \end{aligned}$$

We can write a transfer function bounding the number of iterations N

$$\gamma_{CTRNN}(y_0, t) \approx \sum_{k=0}^{N} y_{k-1} + \Delta t \cdot f(y_0, I)$$

This transfer function considered in a bounded time interval is still o-minimal. Thus also this equation admits a finite abstraction.

Part I

Bibliography

Bibliography

Abbott, L. F., Kepler, T. B., 1990. Model Neurons: From Hodgkin-Huxley to Hopfield. In: Garrido, L. (Ed.), Statistical Mechanics of Neural Networks. XI Sitges Conference, Springer-Verlag, Berlin, pp. 5–18.

Almeida, L. B., 1990. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment, 102–111.

Andersen, R., Snyder, L., Bradley, D., Xing, J., 1997. Multimodal representation of space in the posterior parietal cortex and its use in planning movements. Annual review of neuroscience 20, 303–330.

Antoulas, A. C., Sorensen, D. C., Gugercin, S., 2001. A survey of model reduction methods for large-scale systems. Contemporary Mathematics 280, 193–219.

Bailu, Sil, T. A., 2009. The role of competitive learning in the generation of dg fields from ec inputs. Cognitive Neurodynamics 3 (2), 177–187.

Beer, R. D., 1995a. A dynamical systems perspective on agent-environment interaction. Artificial Intelligence 72, 173–215.

Beer, R. D., 1995b. On the dynamics of small continuous-time recurrent neural networks. Adaptive Behavior 3 (4), 469–509.

Beer, R. D., 2006. Parameter space structure of continuous-time recurrent neural networks. Neural Computation 18 (12), 3009–3051.

Bishop, C. M., 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Blynel, J., Floreano, D., 2003. Exploring the T-Maze: Evolving Learning-Like Robot Behaviors using CTRNNs. In: 2nd European Workshop on Evolutionary Robotics (EvoRob'2003).

Branicky, M. S., 1995. Universal computation and other capabilities of hybrid and continuous dynamical systems. Theoretical Computer Science 138 (1), 67–100.

Brihaye, T., 2006. Verification and control of o-minimal hybrid systems and weighted timed automata. Ph.D. thesis, Université de Mons-Hainaut.

Clarke, E. M., Grumberg, O., Peled, D. A., January 2000. Model Checking. The MIT Press.

Cotter, N. E., Conwell, P. R., June 1990. Fixed-weight networks can learn. pp. 553–559 vol.3.

De Falco, I., Della Cioppa, A., Donnarumma, F., Maisto, D., Prevete, R., Tarantino, E., 2008. CTRNN parameter learning using DIfferential Evolution. In: Ghallab, M., Spyropoulos,

C. D., Fakotakis, N., Avouris, N. (Eds.), ECAI 2008, 18th European Conference on Artificial Intelligence. pp. 783–784.

Dehaene, S., 2005. Evolution of Human Cortical Circuits for Reading an Arithmetic: The "Neuronal Recycling" Hypothesis. From Monkey Brain to Human Brain. A Fyssen Foundation Symposium. Bradford MIT Press, Ch. 8, pp. 133–157.

Donnarumma, F., Prevete, R., Trautteur, G., March 2007. Virtuality in neural dynamical systems. In: International Conference on Morphological Computation. Venice Italy.

Dunn, N. A., Lockery, S. R., Pierce-Shimomura, J. T., Conery, J. S., 2004. A neural network model of chemotaxis predicts functions of synaptic connections in the nematode caenorhabditis elegans. Journal of Computational Neuroscience 17 (2), 137–147.

E., Oztop, A. M., 2002. Schema design and implementation of the grasp-related mirror neuron system. Biological Cybernetics 87 (2), 116–140.

Eck, D., 2006. Generating music sequences with an echo state network. In: NIPS 2006 workshop on Echo State Networks and liquid state machines.

Fadiga, L., Fogassi, L., Gallese, V., Rizzolatti, G., 2000. Visuomotor neurons: ambiguity of the discharge or 'motor' perception? International Journal of Psychophysiology 35, 165–177.

Floreano, D., Mondada, F., 1994. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In: Proceedings of the Conference on Simulation of Adaptive Behavior. MIT Press, Cambridge, MA, USA, pp. 421–430.

Fontana, W., 2006. Pulling Strings. Science 314 (5805), 1552–1553.

Friston, K., Kiebel, S., 2009. Cortical circuits for perceptual inference. Neural Networks In Press, Corrected Proof, –.

Funahashi, K., Nakamura, Y., 1993. Approximation of dynamical systems by continuous time recurrent neural networks. Neural Networks 6 (6), 801–806.

Fyhn, M., Hafting, T., Treves, A., Moser, M.-B., Moser, E. I., March 2007. Hippocampal remapping and grid realignment in entorhinal cortex. Nature 446 (7132), 190–194.

Gallese, V., Goldman, A., December 1998. Mirror neurons and the simulation theory of mind-reading. Trends in Cognitive Sciences 2 (12), 493–501.

Garzillo, C., Trautteur, G., 2009. Computational virtuality in biological systems. Theoretical Computer Science 410 (4-5), 323–331.

Girard, A., Pappas, G. J., 2005. Approximate bisimulations for nonlinear dynamical systems. In: Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on. pp. 684–689.

Girard, A., Pappas, G. J., May 2007. Approximation metrics for discrete and continuous systems. Automatic Control, IEEE Transactions on 52 (5), 782–798.

Graça, D. S., Campagnolo, M. L., Buescu, J., 2005. Robust simulations of turing machines with analytic maps and flows. New Computational Paradigms, 167–179.

Guckenheimer, J., Holmes, P., 1986. Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields. Springer-Verlag.

Gupta, M. M., Homma, N., Jin, L., 2003. Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory. John Wiley & Sons, Inc., New York, NY, USA.

Hale, J. K., Koçac, H., 1991. Dynamics and Bifurcations. Springer-Verlag.

Haschke, R., 2004. Input space bifurcation manifolds of recurrent neural networks. Ph.D. thesis, Bielefeld University, Neuroinformatics Group, Faculty of Technology, Bielefeld, Germany.

Haydon, P. G., Carmignoto, G., 2006. Astrocyte control of synaptic transmission and neurovascular coupling. Physiological Reviews 86 (3), 1009–1031.

Hines, M., Carnevale, N. T., 1998. Computer modeling methods for neurons. The Handbook of Brain Theory and Neural Networks, 226–230.

Hochreiter, S., Younger, S. A., Conwell, P. R., 2001. Learning to learn using gradient descent. In: ICANN '01: Proceedings of the International Conference on Artificial Neural Networks. Springer-Verlag, London, UK, pp. 87–94.

Hopfield, J. J., Tank, D. W., 1985. Neural computation of decisions in optimization problems. Biological Cybernetics 52 (3), 141–152.

Hopfield, J. J., Tank, D. W., 1986. Computing with neural circuits: A model. Science 233, 625–633.

Huo, J., Murray, A., 2009. The adaptation of visual and auditory integration in the barn owl superior colliculus with spike timing dependent plasticity. Neural Networks 22 (7), 913–921.

Ito, M., Tani, J., 2004. Generalization in learning multiple temporal patterns using RNNPB. In: ICONIP: International Conference on Neural Information Processing. pp. 592–598.

Izhikevich, E. M., 2004. Which model to use for cortical spiking neurons? IEEE Transactions on Neural Networks 15 (5), 1063–1070.

Izquierdo-Torres, E., Harvey, I., 2007. Hebbian learning using fixed weight evolved dynamical 'neural' networks. IEEE Symposium on Artificial Life, ALIFE, 394–401.

Izquierdo-Torres, E., Harvey, I., Beer, R. D., 2008. Associative learning on a continuum in evolved dynamical neural networks. Adaptive Behavior 16, 361–384.

Jaeger, H., Lukosevicius, M., Popovici, D., Siewert, U., 2007. Optimization and applications of echo state networks with leaky- integrator neurons. Neural Networks 20 (3), 335–352.

Kambhampati, C., Garces, F., Warwick, K., 2000. Approximation of non-autonomous dynamic systems by continuous time recurrent neural networks. Neural Networks, IEEE - INNS - ENNS International Joint Conference on 1, 1064.

Khalil, H. K., December 2002. Nonlinear systems. Prentice Hall.

Kiebel, S. J., Daunizeau, J., Friston, K. J., 11 2008. A hierarchy of time-scales and the brain. PLoS Comput Biol

4 (11), e1000209.
URL http://dx.doi.org/10.1371%2Fjournal.pcbi.1000209

Kier, R. J., Ames, J. C., Beer, R. D., Harrison, R. R., 2006. Design and implementation of multipattern generators in analog vlsi. IEEE Transactions on Neural Networks 17 (4), 1025–1038.

Koenig, N., Howard, A., September 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. Sendai, Japan, pp. 2149–2154.

La Camera, G., Rauch, A., Thurbon, D., Luscher, H.-R., Senn, W., Fusi, S., 2006. Multiple Time Scales of Temporal Response in Pyramidal and Fast Spiking Cortical Neurons. J Neurophysiol 96 (6), 3448–3464.

Lambrinos, D., Möller, R., Labhart, T., Pfeifer, R., Wehner, R., 2000. A mobile robot employing insect strategies for navigation. Robotics and Autonomous Systems 30, 39–64.

Lang, B., 2005. Monotonic multi-layer perceptron networks as universal approximators. In: Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005. pp. 31–37.

Lapedes, A., Farber, R., 1986. A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition. Phys. D 2 (1-3), 247–259.

Maass, W., Natschläger, T., Markram, H., 2002. Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Computation 14 (11), 2531–2560.

Marker, D., 2002. Model Theory: an introduction. Springer.

McCulloch, W. S., Pitts, W. H., 1943. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysic 5, 115–133.

Nishide, S., Ogata, T., Yokoya, R., Tani, J., Komatani, K., Okuno, H. G., 2009. Autonomous motion generation based on reliable predictability. Journal of Robotics and Mechatronics 21 (4).

Paine, R. W., Tani, J., 2004. Motor primitive and sequence self-organization in a hierarchical recurrent neural network. Neural Networks 17 (8-9), 1291–1309, new Developments in Self-Organizing Systems.

Parrilo, P. A., December 2003. Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization. Ph.D. thesis, California Institute of Technology, Pasadena, CA.

Pearlmutter, B. A., September 1995. Gradient calculations for dynamic recurrent neural networks: A survey. IEEE Transactions on Neural Networks 6 (5), 1212–1228.

Pena, J., Konishi, M., 2004. Robustness of multiplicative processes in auditory spatial tuning. The Journal of Neuroscience 24 (40), 8907–8910.

Pineda, J. A., 1987. Generalization of back propagation to recurrent and higher order neural networks. In: NIPS. pp. 602–611.

Prajna, S., Papachristodoulou, A., Parrilo, P. A., 2002. Introducing sostools: a general purpose sum of squares program-

ming solver. In: Decision and Control, 2002, Proceedings of the 41st IEEE Conference on. Vol. 1. pp. 741–746 vol.1.

Price, K. V., Storn, R. M., Lampinen, J. A., 2005. Differential Evolution: A Practical Approach to Global Optimization. Natural Computing Series. Springer-Verlag.

Reeve, R., Webb, B., Horchler, A., Indiveri, G., Quinn, R., 2005. New technologies for testing a model of cricket phonotaxis on an outdoor robot. Robotics and Autonomous Systems 51 (1), 41–54.

Regev, A., Shapiro, E., 2002. Cellular abstractions: Cells as computation. Nature 419 (6905), 343.

Riesenhuber, M., Poggio, T., 2002. Neural mechanisms of object recognition. Current Opinion in Neurobiology 12 (2), 162–168.

Rizzolatti, G., Gentilucci, M., 1988. Motor and visual-motor functions of the premotor cortex. Neurobiology of Neocortex, 269–284.

Rumelhart, D., Hinton, G., McClelland, J., 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1. MIT Press Cambridge, MA, USA, pp. 605–636.

Salinas, E., Abbott, L., 1996. A model of multiplicative neural responses in parietal cortex. In: PNAS. Vol. 93. pp. 11956–11961.

Salmen, M., Plöger, P. G., April 2005. Echo state networks used for motor control. In: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on. pp. 1953–1958.

Schindler, K., Gool, L. V., de Gelder, B., 2008. Recognizing emotions expressed by body pose: A biologically inspired neural model. Neural Networks 21 (9), 1238–1246.

Shannon, C., 1956. A universal turing machine with two internal states. Automata Studies, 157–165.

Siegelmann, H. T., 1999. Neural Networks and Analog Computation Beyond the Turing Limit. Birkäuser.

Siegelmann, H. T., Ben-Hur, A., Fishman, S., 2000. Comments on attractor computation. International Journal of Computing Anticipatory Systems.

Siegelmann, H. T., Sontag, E. D., 1995. On the computational power of neural nets. Journal of Computer and System Sciences 50 (1), 132–150.

Skowronski, M. D., Harris, J. G., 2007. Automatic speech recognition using a predictive echo state network classifier. Neural Networks 20 (3), 414–423.

Sloman, A., 2008. The well-designed young mathematician. Artificial Intelligence 172 (18), 2015–2034.

Sloman, A., Chrisley, R., 2003. Virtual machines and consciousness. Journal of Consciousness Studies 10, 4–5.

Sontag, E. D., 1990. Mathematical control theory: deterministic systems. Springer-Verlag New York, Inc., New York, NY, USA.

Sontag, E. D., 1996. Critical points for least-squares problems involving certain analytic functions, with applications to sigmoidal nets. In: Advances in Computational Mathematics

(Special Issue on Neural Networks. Publications, pp. 245–268.

Steil, J. J., July 2004. Backpropagation-decorrelation: online recurrent learning with o(n) complexity. Vol. 2. pp. 843–848 vol.2.

Strogatz, S. H., 1994. Nonlinear dynamics and chaos. Addison Wesley, New York.

Tabuada, P., Ames, A. D., Julius, A., Pappas, G. J., 2008. Approximate reduction of dynamic systems. Systems & Control Letters 57 (7), 538–545.

Tani, J., Ito, M., Sugita, Y., 2004. Self-organization of distributedly represented multiple behavior schemata in a mirror system: reviews of robot experiments using rnnpb. Neural Networks 18 (1), 103–104.

Tino, P., Horne, B. G., Giles, C. L., 2001. Attractive periodic sets in discrete time recurrent networks (with emphasis on fixed point stability and bifurcations in two-neuron networks). Neural Computation 13, 1379–1414.

Tong, M. H., Bickett, A. D., Christiansen, E. M., Cottrell, G. W., 2007. Learning grammatical structure with echo state networks. Neural Networks 20 (3), 424–432.

Trautteur, G., Tamburrini, G., 2007. A note on discreteness and virtuality in analog computing. Theoretical Computer Science 371, 106–114.

Van Den Dries, L., 1998. Tame Topology and O-minimal Structures. Cambridge University Press.

Yamauchi, B. M., Beer, R. D., 1994. Sequential behavior and learning in evolved dynamical neural networks. Adaptive Behavior 2 (3), 219–246.

Yamazaki, T., Tanaka, S., 2007. The cerebellum as a liquid state machine. Neural Networks 20 (3), 290–297.

Younger, A., Conwell, P., Cotter, N., 1999. Fixed-weight online learning. IEEE Transactions on Neural Networks 10 (2), 272–283.

Younger, A. S., Redd, E., 2008. Learning at the speed of light: A new type of optical neural network. In: Optical SuperComputing. Vol. 5172. Springer Berlin / Heidelberg, pp. 104–114.

Zegers, P., Sundareshan, M. K., May 2003. Trajectory generation and modulation using dynamic neural networks. IEEE Transactions on Neural Networks 14, 520–533.

Zhang, Y., September 1994. A foundation for the design and analysis of robotic systems and behaviors. Ph.D. thesis, University of British Columbia.