INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Relativizations of P = NP and P $\neq$ NP in ARNN Model

Raimundo Coelho Leong

Dissertação para obtenção do Grau de Mestre em
Matemática e Aplicações

Novembro de 2010

To family.

Acknowledgments

On January 2010, I was about to make a decision, one that changed my life.

I still remember clearly. On December 2009, I was having lunch with one colleague of mine in a pizza restaurant. She was asking me about what I would like to study after finishing the Master's degree. This was a conversation copyrighted by me. We were both looking for some interesting applications of Mathematics, something attractive. She told me about Neuroscience. And here started my journey.

I have to thank this friend, Catarina, for giving me this *aha!* moment. For abandoning my previous thesis, I owe an apology and many thanks for the support and patience to my ex-supervisor, Professor Pedro Resende.

For every lesson, every interesting discussion, every help (including some diagrams in this thesis) and this great opportunity to learn about almost everything, accepting me as a thesis student in the area of Computation and Complexity theory on ARNN, a very big thank you to my supervisor Professor José Félix Costa. He helped me on every little detail on this journey, starting from explaining to me what a Turing machine is.

A very big thank you to a group of friends, including colleagues and professors, that walked along the first years of my life in Portugal.

All the company I have in Instituto Superior Técnico, a very big thank you. For those who helped me out on many difficulties, colleagues from "o gabinete" and "a sala de estudo", thank you for all your patience and for listening and giving me important advices. Big lessons I have learnt in these two years of my life. Many thanks also to my friends from the residence for all the funny and relaxing moments. Thank you all my friends from Lisbon.

To all my friends of Macau, widespreaded all over the world, Dó jé nei tei ah! First, my teacher of Mathematics from secondary school, Professora Antónia Costa. I will never forget every break between lessons that I took from you to explain my wild ideas. Thanks to Celina, my "seed" and to my "sister-from-another-mother", Dora. May our friendship grow old with us.

André, you deserve a very special hug, always supporting me, day and night, every single moment.

Last but not least, a big kiss to my family, for the unlimited support without questioning and doubts from the other side of the world and all these years of love and care. My gratitude for you is infinite. Love you all!

Agradecimentos

Em Janeiro de 2010, estava prestes a fazer uma decisão que mudou a minha vida.

Ainda me lembro muito bem. Em Dezembro de 2009, estava a tomar um almoço com uma colega minha numa pizzaria. Ela perguntou-me sobre o que eu queria estudar depois de termintar o Mestrado. Sobre este tema de conversa tenho os direitos de autor como todos os meus amigos sabem. Estávamos ambos à procura de algumas aplicações interessantes da Matemática, algo atraente. Ela falou-me sobre Neurociências. E aí comecei o meu percurso.

Tenho de agradecer esta amiga minha, Catarina, por me dar este momento *aha!* Por abandonar o tema anterior para a tese, devo um pedido de desculpas ao meu ex-orientador Prof. Dr. Pedro Resende. Muito obrigado pelo suporte e pela paciência.

Por cada lição, cada discussão interessante, cada ajuda (incluindo alguns diagramas que constam nesta tese) e por esta grande oportunidade de aprender sobre quase tudo, aceitando-me como orientando na área de Computação e Complexidade em ARNN, muito obrigado ao meu orientador Prof. Dr. José Félix Costa. Ele ajudou-me em cada detalhe neste percurso, desde explicar-me o que é uma máquina de Turing.

Agradeço imenso ao grupo de amigos, incluindo colegas e professores, que caminharam comigo nos primeiros anos da minha vida em Portugal.

A todos os companheiros do Instituto Superior Técnico, muito obrigado. Àqueles que me ajudaram em várias situações de dificuldades, colegas do "gabinete" e da "sala de estudo", um grande agradecimento pela vossa paciência, por me ouvir e por me dar conselhos quando mais precisei. Muito aprendi com todos nestes dois anos da minha vida. Obrigado também aos meus colegas de residência por todos os momentos de diversão e de relaxamento. Obrigado a todos meus amigos de Lisboa.

Aos meus amigos de Macau espalhados por todos os cantos do mundo, Dó jé nei tei ah! Primeiro, à minha professora de Matemática da escola secundária, Professora Antónia Costa. Nunca me esquecerei dos intervalos que te roubei para explicar as minhas ideias loucas. Obrigado Celina, a minha "semente" e um abraço grande à minha "sister-from-another-mother", Dora. Que a nossa amizade dure para sempre.

André, mereces um abraço muito especial, pelo suporte constante, dia e noite, em cada instante.

Por fim, um beijo enorme à minha família, pelo suporte sem questionar e duvidar vindo do outro lado do mundo e por estes anos todos de amor e carinho. Amo-vos a todos!

Abstract

We will provide a formalization of the use of real weights in analog recurrent neural networks as a restricted class of oracles, closely related to physical oracles, and prove that the famous $P = NP$ problem relativizes with respect to deterministic and nondeterministic analog recurrent neural networks with real weights working in polynomial time. As a direct consequence, these devices have restricted computational power.

Keywords

- Analog Recurrent Neural Networks
- Davis's Oracles
- Real weights in ARNN
- Relativization in ARNN

Resumo

Apresentamos uma formalização da implementação de pesos reais em redes neuronais como uma classe restrita de oráculos fortemente relacionados com oráculos físicos e provamos que a famosa hipótese $P = NP$ relativiza no caso de redes neurais determinísticas e não-determinísticas com pesos reais com tempo de computação polinomial. Como consequência, estes sistemas revelam ter um poder computacional restrito.

Palavras-chave

- Redes Neuronais
- Oráculos de Davis
- Pesos reais em ARNN
- Relativização em ARNN

Contents

Preface

Gandy once referred in [8] to a result he called the Turing's Theorem which states that

Any function which is effectively calculable by an abstract human being following a fixed routine is effectively calculable in the sense defined by Church – and conversely.

Human being has always been interested in understanding how abstract reasoning is generated in our brains and between them. Great efforts have been made from different areas such as psychology, physiology and biology. In the computational sense, the first big step was to formalize a concept reasonably strict but still sufficiently wide of what is calculable. In the well known year of 1936, different ideas of effective calculability arised which have later been proved to be equivalent between them: the Church-Turing Thesis. As a result of this confluence of ideas, Computability Theory was born.

Many attempts have been made to create models of the human mind and important philosophical discussions arose around this subject. Warren McCulloch and Walter Pitts in [11] first presented a model of threshold logic units, forming what is called the neural networks. These have been proved by Kleene to be equivalent to abstract devices called finite automata. Threshold units together with a Heaviside function as activation can hold binary values only. This model has been strongly criticized during the Macy's Conferences for its lack of biological and neurophysiological plausibility (see [13]) but they surely satisfy the requirements of a computational device: the inputs are fed in in binary fashion; the output streams are encoded in binary; the system has a finite dimension, corresponding to a finite control.

Neural networks with rational weights and a piecewise linear activation function were widely used. As the rational numbers can encode information of arbitrary (but finite) size, for example, in Cantor subsets of the interval $[0, 1]$ ([16], [17]), units holding rational values resemble infinite tapes in Turing machines. In fact, they have been proven by Hava Siegelmann and Eduardo Sontag in [17] to be equivalent and, moreover, that there is a universal architecture of rational neural networks that simulates a given Turing machine in real time.

In further research, neural networks carrying real values were shown to have the same computational power as families of Boolean circuits ([16]). If no time bound is set, they can decide any set! And when a polynomial time bound is imposed, they decide exactly $P/poly$. This idea of "surpassing" the power of a Turing machine has attracted the attention of Jack Copeland ([6]). Martin Davis strongly criticized this by recalling that these real weights are in fact oracles, as they were implemented into the network along with their non-computability. These oracles still have to be formalized. It seems that no one noticed that these oracles are related to the $P = NP$ problem, that is, the relativization problems.

Our goal is to define these oracles, we call them Davis's oracles in memory of his criticism, and show that Turing machines equipped with these oracles can be simulated by a neural network with real weights and conversely. It is known that $P = NP$ relativizes for the physical oracles [5]. We will regard them as a particular case of Davis's oracles and show that $P = NP$ relativizes also for this larger class, proving them to be a strict class of oracles.

We will be first directing in Part I to some basic notions and classical results about neural networks. In Chapter 1, some ideas from neurobiology of how the nervous system works will be introduced.

A formal theory of systems will be presented in Chapter 2 to make possible the definition of a neural network. Softwired networks will be briefly mentioned, as we will mainly be addressing results on hardwired ones (for further readings about softwired networks, see [1]).

In Chapter 3, some computational issues such as combination of neural networks in sequence and in parallel and synchronization will be discussed. The main idea of this chapter is to make these architectures and their dynamics familiar to the reader.

Deterministic neural networks with integer and rational weights are presented and proved to decide respectively regular and recognizable languages in Chapter 4. Here, we provide a sketch of the proof of real time simulation of Turing machines by rational neural nets. This result will be essential for the second part of this work. We will briefly explore non-deterministic neural networks.

Along Part II, a path to relativization results will be drawn. In Chapter 1, we will recall the definition of o-machines and formalize the idea of a Davis's oracle. The circuit value problem, strongly related to the implementation of a real weight, will be mentioned and some results will be proved. A subsystem called BAM will be emphasized as an important part of the proof of the simulation of real networks by Turing machines querying Davis's oracles. Time protocols of oracles will be introduced and Davis's oracles with polynomial and exponential access time will be defined.

In Chapters 2 and 3, we will show relativization results regarding Davis's oracles with polynomial and exponential access time. Scatter machine experiments will be related to real weights with polynomial access time, both included in the class of Davis's oracles with polynomial access time. Collider machine experiments will be similarly related to those with exponential access time. In the first case, we will prove that $P = NP$ positively relativizes and in the second both positively and negatively. These are the main results of this project.

As James Anderson refers in [1]:

We hope to show that there is no magic in neural networks. Networks do suggest, however, a number of fascinating and useful ideas that in the long run are of more value than magic.

Part I

ARNN

Chapter 1

Neurobiology

The nervous system is mainly a network of *neurons* or *nerve cells* responsable for body movements' coordination and information transmission between different parts of the body.[1] In human, there are apparently about 10^{10} to 10^{11} neurons, each connected to hundreds to thousands of others, totalling 10^{14} to 10^{15} of connections. These cells are highly sensitive to pressure and metabolically very active (they consume a substantial part of the body's energy to maintain their function). In mammals, the self-dividing process of the neurons stops after birth.[2]

Each neuron is composed by: *soma*, its cell body; *dendrites*, a tree-shaped extension; and a special extension called *axon*. The signal transmission, chemical or electrical, between neurons occurs between axon and dendrites, in the *synapse* (see Fig. 1.1). As a signal is fired by a neuron's soma, runs through the axon arriving to the *terminal arborization*. In the synapse, this signal is transmitted to another neuron's soma through dendrites.

The nerve cell has a membrane mainly composed by lipids and proteins, whose permiability to ion flow through ionic channels can be controlled by electrical and chemical environment. Electrical signals are then measurements of the potential in the membrane of the inside of the cell relatively to the environment surrounding it. The ions responsable for it are Na^+ and K^+. To hold the proportions of the concentration of these ions on both sides of the membrane in an equilibrium (when the neuron is not excited), the membrane itself should keep -60 mV of voltage, the so-called *membrane potential in rest*. This value can be approximated by the Nernst Equation and corresponds to the equilibrium of K^+:

$$E = \frac{RT}{F} ln \frac{c_1}{c_2}$$

where F is the Faraday constant, R is the gas constant, T the temperature of the ion (thought as an ideal gas) and c_1 and c_2 the concentration of the ion on both sides of the membrane.

The membrane has an associated *threshold* potential responsable for whether the soma fires a signal or not. When a membrane is depolarized, that is, when its potential is increased by inducing a current[3], either one of the two situations occurs. The potential

[1] There are many other cells composing the nervous system, such as the *glia*. We will omit them since we consider them computationally irrelevant.

[2] This explains why we consider systems of fixed dimensions.

[3] Through a microelectrode for example.

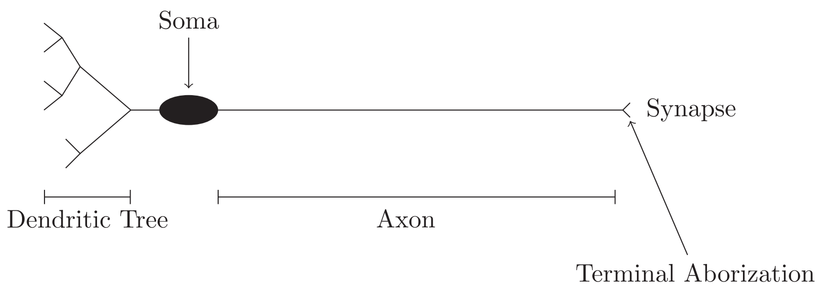


Figure 1.1: An idealized neuron

induced is not enough to surpass the threshold and the membrane returns to the equilibrium of K^+. Otherwise a sudden *spike* – an *action potential* – is generated by the membrane, taking its potential to some tens of milivolts corresponding to the membrane potential of the equilibrium of Na^+. Immediately after, the threshold increases, making it more difficult to excite.[4]

The spike is only generated if the membrane potential is greater than the threshold so it works in all-or-none fashion and it's shape does not change with increasing current once it is generated. These properties resemble those of a true-false valued unit, allowing us to create a computational model for neural networks. In the next chapters we will present a model of units with fixed thresholds introduced by Warren McCulloch and Walter Pitts in [11] based on these properties.

When more than one signal is transmitted along a short period of time from one cell, *presynaptic neuron*, to another, *postsynaptic neuron*, the membrane of the dendrites of the postsynaptic neuron works as an analog computer, refining and summing up the input signals, i.e., it performs *time integration*.[5] This integrating processing function seems to be related to the shape of the dendritic tree. If we change the morphology of a *dendritic spine*, part of the postsynaptic cell where synapse occurs, the synaptic strength of the presynaptical cell over the postsynaptic one also changes and this can happen independently of other spines[6]. Softwired nets suggest this plasticity of synaptic strengths, permitting learning processes to be modelled.

Spatial integration in one cell, that is, the interaction of synapses coming from different neurons, also occurs.

[4]A neuron can then be seen as a voltage-frequency converter.

[5]Although linear integration is not the general case, this phenomenon still happens in some simple cases such as the well studied eye of the horseshoe crab.

[6]Around a dendritic spine, there is an "isolated biochemical microenvironment".

As for an introduction of simple computational models of the nervous system, we have to abandon some important properties such as the noise of the signals, synaptic delays and *refractory periods*, corresponding respectively to the increasing of the firing threshold after an action potential (relative) or even absolute short-term inhibition of that cell (absolute).

Chapter 2

Formal model

After some insights in basic Neurobiology from the last chapter, we could start a discussion about the biological plausibility of existing computational models of the brain. This is not our goal. We aim to define a theoretical support for these models and then classify them into two kinds of systems: the *softwired* ones – those that admit structural changes after definition – and *hardwired* ones – those that does not.

2.1 General system theory

We will introduce some basic notions to make possible the definition of a general system in a natural fashion. This will be consistent with the one given by Sontag in his famous book (see [18]).

Definition 2.1.1 $(T, \leq, +)$ *is a* ***time set*** *if it is a totally ordered monoid, that is:*

(i) (Antisymmetry) $a \leq b$ *and* $b \leq a$ *implies* $a = b$*;*

(ii) (Transitivity) $a \leq b$ *and* $b \leq c$ *implies* $a \leq c$*;*

(iii) (Totality) $a \leq b$ *or* $b \leq a$*;*

(iv) (Associativity) $(a + b) + c = a + (b + c)$*;*

(v) (Identity) there is an element 0 *such that* $a + 0 = 0 + a = a$

for all $a, b, c \in T$.

Example $\mathbb{R}$, $\mathbb{Z}$

The properties (i), (ii) and (iii) refers to the definition of *totally ordered set* and (iv) and (v) of *monoid.* The monoid operation is important to be able to perform translation in time.

Definition 2.1.2 *An* ***interval*** *is a subset of* T *of the form* $[a, b[= \{c \in T : a \leq c < b\}$.

Two intervals I_1, I_2 are *concatenable* if their disjoint union, written as $I_1 \coprod I_2$, is still an interval. For T a time set, we denote by $\mathcal{T}$ the set of all of its intervals.

Definition 2.1.3 *A* ***path*** *in an arbitrary set U (parametrized by an interval of T) is a function $\omega : I_\omega \in \mathcal{T} \to U$. The empty path is denoted by λ.*

Two paths $\omega_1 : I_1 \to U$, $\omega_2 : I_2 \to U$ are *concatenable* if I_1 and I_2 are concatenable. The result is written as $\omega_2\omega_1 : I_1 \coprod I_2 \to U$ (if all elements of I_1 are smaller than those of I_2).[1]

Ω is closed under restriction in $\mathcal{T}$, that is, if ω is a path, then any of its restriction I' to a subinterval of I_ω, $\omega_| : I' \to U$, is still a path.

Now, we are ready to define a system.

Definition 2.1.4 *Let T be a time set, X, U sets (of* ***states*** *and of* ***values****, respectively) and Ω the set of all paths in U.*

$\Sigma = \langle T, X, U, \Phi \rangle$ is called a ***system*** *if the transition function $\Phi : D_\Phi \subseteq \Omega \times X \to X$ verifies the following conditions:*

- *for all $x \in X$, $\Omega(x) = \{\omega \in \Omega : (\omega, x) \in D_\Phi\}$ is a subset*[2] *with the empty path of U and $\Phi(\lambda, x) = x$;*
- *if (ω_1, x), $(\omega_2, \Phi(\omega_1, x)) \in D_\Phi$ and ω_1, ω_2 are concatenable, then $(\omega_2\omega_1, x) \in D_\Phi$ and $\Phi(\omega_2, \Phi(\omega_1, x)) = \Phi(\omega_2\omega_1, x)$.*

An element of D_Φ is an *admissible pair* and a path ω is *admissible* if there is $x \in X$ such that $(\omega, x) \in D_\Phi$.

2.2 An overview on Dynamical Systems

Definition 2.2.1 *A* ***dynamical system*** *is a tuple $\Sigma = \langle T, X, \Psi \rangle$ where T is a time set, X a set and $\Psi : U \subseteq \mathcal{T} \times X \to X$ such that for all $x \in X$:*

- *$(\emptyset, x) \in U$;*
- *if $T(x) = \{I \in \mathcal{T} : (I, x) \in U\}$, then*

$$\Psi(\emptyset, x) = x$$

  and

$$\Psi(I_2, \Psi(I_1, x)) = \Psi\left(I_1 \coprod I_2, x\right)$$

  for all I_1, I_2 concatenable intervals in $T(x)$.[3]

We will require another property:

- $(I, x) \in U$ implies $(I', x) \in U$ for all subinterval I' of I.

[1]The set Ω of all paths is a category with the operation of concatenation of paths.

[2]A subcategory...

[3]This definition of dynamical system is more general than the classical one, since here not only the size of the time-step is considered, but also the initial instant.

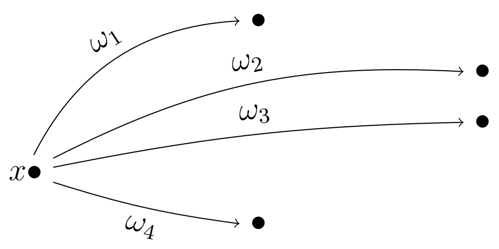
Let $\Sigma = \langle T, X, U, \Phi \rangle$ be a system. Fix $\omega : I_\omega \to U$ in Ω and

$$U(\omega) = \left\{ (I, x) \in \mathcal{T} \times X : I \subseteq I_\omega, \left(\omega_{|I}, x\right) \in D_\Phi \right\}.$$

Then $\Phi_\omega : U(\omega) \to X$ defines a dynamical system.

Therefore, a system can be regarded as a family of dynamical systems, indexed by the admissible paths in U.

If, instead, we fix a state $x \in X$, $\Phi_x : \Omega(x) \to X$ can then be interpreted as a collection of admissible "future states" with the initial state x.



2.3 Discrete systems

Definition 2.3.1 *A system is **discrete** if $T = \mathbb{Z}$.*

In discrete systems, paths can be "atomized". We can write them as a concatenation of paths of one time step $\omega_t : \{t\} \to U$. We will use $u : \{t\} \to U$ to denote these "atomized" paths where $u(t) = u \in U$. In fact, an "atomized" path is not more than a choice of a pair (t, u) in $T \times U$.

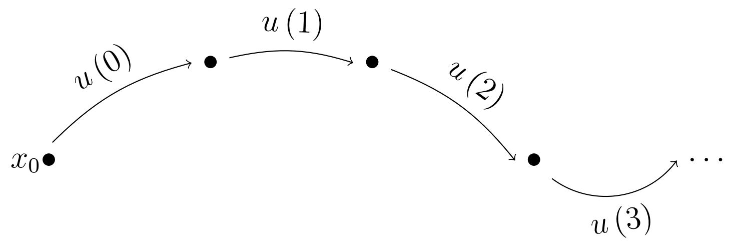
Definition 2.3.2 *The **dynamic map** of a discrete system $\Sigma = \langle \mathbb{Z}, X, U, \Phi \rangle$ is a family of functions $\{F_t : D_t \subseteq U \times X \to X\}_{t \in \mathbb{Z}}$ verifying the commutative diagram below*

$$\begin{tikzcd}
(u(t), x) \arrow[r, mapsto, "F_t"] & x^+ \in X \\
(u, x) \arrow[u, mapsto, "(t, id)^\star"] \arrow[ur, mapsto, "\Phi"] &
\end{tikzcd}$$

Note that from the last definition we can conclude that giving the dynamic map of a discrete system is equivalent to giving its transition function.

After fixing initial conditions $x(t_0) = x_0$ for time and state, we can write $x(t+1) = x^+ = F_t(u(t), x(t))$, that is, given a path of values $(u(t))_t$ (which will be seen as input) and initial conditions, the result after iteration by F is a path of states.[4]

[4]From now on, we will adopt the notation of x^+ for $x(t+1)$ except when needed.



Definition 2.3.3 *A system $\Sigma = \langle T, X, U, \Phi \rangle$ is **complete** if $D_\Phi = \Omega \times X$.*

Definition 2.3.4 *$\Sigma = \langle \mathbb{Z}, X, U, \Phi \rangle$ is a **semilinear discrete system** (over $\mathbb{R}$) if:*

- *Σ is complete;*
- *X and U are vector spaces (over $\mathbb{R}$);*
- *$F = \sigma \circ \pi$, where π is an affine map and σ, called the **activation function**, a vector of non-linear functions ($\mathbb{R} \to \mathbb{R}$)*

The *dimension* of such system is the dimension of X. Σ is said to be of *finite dimension* if the dimensions of X and U are finite. In this case, $x^+ = \sigma^t \left(A(t)\, x(t) + B(t)\, u(t)\right)$ where $A(t) : X \to X$ and $B(t) : U \to X$ are affine maps.

Definition 2.3.5 *A system is **time invariant** if Φ is invariant by translation in time, i.e., for all $(\omega, x) \in D_\Phi$, $(\omega^\mu, x) \in D_\Phi$, where*

$$\omega^\mu(t) = \omega(t - \mu),$$

and

$$\Phi(\omega, x) = \Phi(\omega^\mu, x).$$

Thus the dynamic map of a time invariant system does not depend explicitly on time. In particular, in a time invariant semilinear discrete system we have

$$x^+ = \sigma\left(Ax(t) + Bu(t)\right)$$

with A and B affine or equivalently,

$$x^+ = \sigma\left(Ax(t) + Bu(t) + c\right)$$

with A and B linear maps and c a vector of X. The entries of the matrix representing A, B and c are called *weights*.

2.4 The neural net case

Now, consider finite dimension, discrete time, semilinear systems.

The time set is clearly the monoid $\mathbb{Z}$.

The vector space X is $\mathbb{R}^n$, for some specified n, although we will also consider subsets, namely $\mathbb{Q}^n$ (also as a subspace) and $\mathbb{Z}^n$. The *state variables*, or *units*, will be denoted by $\vec{x}$ of n components $x_1, \ldots, x_n$.

The vector space U is $\mathbb{R}^m$, for some m, although we will consider in what follows Boolean vectors. *Inputs* will be total functions from $\mathbb{N} \to \{0,1\}$, i.e., streams of Boolean values. The *state of the input* is given at any moment of time t by a vector $\vec{u}(t)$ of m components $u_1(t)$, ..., $u_m(t)$.

To specify the dynamical map we will consider $\sigma_1, \ldots, \sigma_n$ independent of time and matrices $A(t)$, of dimension $n \times n$ and $B(t)$ of dimension $n \times m$, both composed by real numbers. Sometimes we will restrict those values to the rationals or the integers. We can always consider a state variable with fixed value 1 and workout the dynamic map to write it as follows:

$$\begin{cases} x_1^+ = \sigma_1(a_{11}(t)x_1(t) + \cdots + a_{1n}(t)x_n(t) + b_{11}(t)u_1(t) + \cdots + b_{1m}(t)u_m(t) + c_1(t)) \\ \vdots \\ x_n^+ = \sigma_n(a_{n1}(t)x_1(t) + \cdots + a_{nn}(t)x_n(t) + b_{n1}(t)u_1(t) + \cdots + b_{nm}(t)u_m(t) + c_n(t)) \end{cases}$$

This system can be presented in abbreviated form by $\vec{x}^+ = \vec{\sigma}(A(t)\vec{x}(t) + B(t)\vec{u}(t) + \vec{c}(t))$. The most common functions used as $\sigma_1, ..., \sigma_n : \mathbb{R} \to \mathbb{R}$ are within the following classes:

(a) The *McCulloch-Pitts sigmoid* (see [9], [11]),

$$\sigma_d(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

(b) The *saturated sigmoid*, used by Siegelmann and Sontag in the nineties,

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

(c) The *analytic sigmoid* of parameter k (see [9]),

$$\sigma_a^k(x) = \frac{2}{1 + e^{-kx}} - 1$$

Definition 2.4.1 *Given a system (Σ) $\vec{x}^+ = \vec{\sigma}(A(t)\vec{x}(t) + B(t)\vec{u}(t) + \vec{c}(t))$, with initial condition $\vec{x}(0)$ and $\vec{u}(0)$, a* ***finite computation*** *of Σ is a sequence of state transitions $\vec{x}(0)\vec{x}(1)...\vec{x}(t)$ such that, for every $1 \leq \tau \leq t$, $\vec{x}(\tau) = \vec{\sigma}(A(\tau-1)\vec{x}(\tau-1) + B(\tau-1)\vec{u}(\tau-1) + \vec{c}(\tau-1))$.*

Definition 2.4.2 *Given a system* (Σ) $\vec{x}^+ = \vec{\sigma}(A(t)\vec{x}(t) + B(t)\vec{u}(t) + \vec{c}(t))$, *with initial condition* $\vec{x}(0)$ *and* $\vec{u}(0)$, *a* ***computation*** *of* Σ *is an infinite sequence of state transitions* $\vec{x}(0)\vec{x}(1)...\vec{x}(t)...$ *such that, for every* $\tau > 0$, $\vec{x}(\tau) = \vec{\sigma}(A(\tau-1)\vec{x}(\tau-1) + B(\tau-1)\vec{u}(\tau-1) + \vec{c}(\tau-1))$.

We choose a collection of state components within the n components to denote the output of the system. Those variables are called *effectors*, provided that they are always Boolean valued. For those effectors we can define an *output stream*, i.e., a map $v : \mathbb{N} \to \{0,1\}$, such that, if x_k is an effector, then $v(t) = x_k(t)$.

The dimensions of X and U are fixed. This allows us to build an architecture that realizes the system. This architecture is composed by m input lines or *sensors*, receiving as *input stream* the states of input, n *neurons*, carying the values of the state variables and equipped with evaluation functions σ, and a collection of *effector neurons* chosen to output the Boolean stream v. These components are denoted by the same letters that denote the values they hold but in capital letters.

This computational structure is a connection between the formal model and the biological model. On one hand, it is a realization of such dynamical systems. On the other, it is an analog resemblance of biological neural networks. Here relies the tough discussions of biological plausibility of these models and the utility and computational power of some machines built from units with their properties.

2.5 Softwiring and Learning Processes

Neural nets were originally inspired by neural and biological characteristics of the brain. As it is believed that long term changes in the synaptic strength is the key process of memory, through algorithms of learning processes, supervised or unsupervised, one expects to teach a finite list of patterns to a neural net by changing the weight matrices and afterwards recall saved information with success. Here, we will do a small deviation to briefly introduce the history of the development of these methods.

The first introduced and most basic learning rules are the famous *Hebbian rules.* It consists on an autoassociative system, an architecture composed by n input lines fully connected to n neurons. Being it a *supervised learning process*, it learns by showing a list of vectors and the weights are corrected by comparing the final output to the original input.

In 1959, Rosenblatt, computer scientist, introduced a system called *Perceptron*, applying the Hebbian learning rules refered in the last assignment. The first perceptron based computer was built in 1960, the famous Mark 1. Being this a breathtaking advance in computer science, many companies sponsored investigation on these systems. In 1969 Minsky and Pappert pointed out some flaws on the idea that perceptron is capable to learn everything. Some simple counterexamples were given, such as the logical operation XOR. Although in terms of computer science the interest on it dimmed out for some years, for psychologists it was still used as a model with much interest, not only in what it is capable, but also its limitations.

The *Widrow-Hoff learning process*, based on the least mean squares (LMS), was introduced in 1960. In the same year, a 3-terminal circuit element that realizes it, called

Memistor, was introduced by Widrow and a neural network architecture called the *ADALINE* (ADAptive LInear NEuron or ADAptive LINear Element) based on these elements was presented by Widrow and Hoff. The main idea is to apply the gradient descent algorithm to minimize the error as much as possible.

In 1982, John Hopfield published an article where some ideas from discrete dynamical systems were introduced as techniques of memory recalling. The weights are corrected by *asynchronous updating*, that is, the weights being altered are randomly chosen given an initially fixed distribuition.

In mid-80's, *Backpropagation*, another application of the gradient descent method, was discovered by several different research groups (David Parker in 1985, Yann Le Cun in 1986, Rumelhart, Hinton and Williams in 1986). It consists on a multiple-layered feedforward net with nonlinear activation functions. These systems are known to solve problems that are not linearly separable, including XOR, so their computational power is higher than simple architectures such as perceptron and the Widrow-Hoff nets. Despite this advantage and its broad applicability – in programs like NETtalk, a net that learns to pronounce English words, and character recognition –, there are still many problems, especially with the interpretation of how it works, which slows down its development.

In opposite to these *supervised learning methods*, one can learn by accumulation of experience, that is, concepts are naturally formed following a given rule without any preconceived pairing between the stimulus and its representation. We call this an *unsupervised learning process*. A set of models called *ART* (adaptive ressonance theory) has been built, being the first one, ART 1, presented by Carpenter and Grossberg in 1987. The basic algorithm follows from the study of stationary conditions of the differential equations that rules the components of the neurons, while the inputs or stimuli presented as binary vectors are analysed and classified by *concepts*[5] The quality of the concepts is controled by a constant associated to the network called the *vigilance parameter*[6]. This is an important component of the structure for a biologically more plausible model.

These methods were intended to mimic biological brain processes or to design more powerful and efficient models. Conversely, they also played or still play an important roll for the understanding and the functional characterization of the brain. For further readings, see [1].

2.6 Hardwiring and ARNN

From now on, consider hardwired structures, that is, the weights of neural nets are fixed at start. This means that we will be dealing with time invariant systems. The dynamics can then be expressed as

$$x^{+} = \sigma\left(Ax\left(t\right) + Bu\left(t\right) + c\right)$$

[5]A concept is a representative element of a cluster of vectors experienced during the learning process.

[6]This parameter resembles the vigilance of visual learning process. If it is low, then new patterns presented are not classified with enough detail and the result would be a poor collection of very few concepts. But if this parameter is very high, then every new pattern is distinct, creating too many undesired concepts.

where A and B are linear maps, represented by matrices[7] and c a vector. The architectures supporting these systems are called *analog recurrent neural networks*, abbreviated as ARNN. The class of these abstract structure will be denoted ARNN.

It is of our interest to study some special classes of ARNN. Restricting the weights to the set of reals, rationals and integers, we have the classes of real, rational and integral ARNN respectively.

Let us display some examples that suggest some computational power of neural nets. For the purpose, we will consider only saturated sigmoids as activation function of the state components of the dynamical system. Some conventions on how to input data and extract the result from these systems have to be established. Let $\phi : \{0,1\}^+ \to \{0,1\}^+$ be a function and α a binary word given as input: we will consider two input streams, one is $0\alpha 0^\omega$ and the other, to validate the sequence of time steps that the input takes, is $01^{|\alpha|}0^\omega$. Analogously, we use two streams for the system to output the result: the first one is the validation line $00^{t-1}1^{|\phi(\alpha)|}0^\omega$, where $t > 1$ is the time step of the first bit of the output, and the output stream $00^{t-1}\phi(\alpha)0^\omega$, where $\phi(\alpha)$ is the expected answer.

By convention, we adopt the initial state $\vec{0}$ at time $t = 0$ and that the input bits at time $t = 0$ are 0. Consequently, at time $t = 1$, the state is $\vec{x}(1) = \sigma(A\vec{x}(0) + B\vec{u}(0) + \vec{c}) = \sigma(\vec{c})$.

Example The first example is a rather simple but clarifying one - the *unary succesor*[8]. There are many networks simulating this operation. In the following paragraphs we will show two of them[9].

Consider the system below:

$$\begin{cases} y_1^+ = \sigma(a) \\ y_{a+}^+ = \sigma(a + y_1) \\ y_v^+ = \sigma(a + y_1) \end{cases}$$

with a as input, y_{a+} as output and y_v its validation. Note that in this case the validation of the input, denoted by v, is not necessary since the input is in unary. The reader can easily check that this system in fact computes the successor of the input in constant time.

Given an input word, for instance 11, we can present the simulation of the associated run by a table as follows:

t	a	y_1	y_{a+}	y_v
0	0	0	0	0
1	1	0	0	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	0

[7]We will denote these matrices by the same letter.

[8]The representation of a natural number in unary is a sequence of 1's, as many as the number represented.

[9]By the way, this kind of examples is not published elsewhere.

Another way to simulate this operation is by considering this system

$$\begin{cases} y_1^+ = \sigma(v) \\ y_{a+}^+ = \sigma(v + y_1) \\ y_v^+ = \sigma(v + y_1) \end{cases}$$

This is almost the same as the former. If the input represents a natural number in unary, the output will be exactly the same in both.[10]

Example Now, we will show a system that computes unary sum in constant time. Here, a, b and y_{a+b} will denote the summands and the output, respectively.

The first part

$$\begin{cases} y_1^+ = \sigma(v) \\ y_2^+ = \sigma(y_1) \end{cases}$$

simply prints out the input of bigger value, digit by digit (being it delayed to match the timing of the next procedure). The second part

$$\begin{cases} y_3^+ = \sigma(a + b - 1) \\ y_4^+ = \sigma(\frac{1}{2}(y_3 + y_4) - (1 - y_3)) \end{cases}$$

codifies the summand with smaller value in *2-Cantor system* (discussed in Appendix B). Next, this result will be "saved" in the next neuron

$$\left\{ \; y_5^+ = \sigma(y_4 + y_5 - (1 - y_1) - y_3) \right.$$

while the result from the first part is being exported. When it ends, the following part decodes the "saved" information [11]:

$$\begin{cases} y_6^+ = \sigma(2(y_5 + y_6 + y_4) - 1 - y_1) \\ y_7^+ = \sigma(2(y_5 + y_6 + y_4) - 2y_1) \end{cases}$$

Finally, the output neuron prints out the result

$$\left\{ \; y_{a+b}^+ = \sigma(y_2 + y_7) \right.$$

This example shows that these systems not only do direct computations, but also have the capacity of memory. This is due to the use of the saturated sigmoid, which allows us to encode uniquely strings of input into a real number (in the last example, into a rational number) in the interval $[0, 1]$. The system of the decoding units, y_6 and y_7, is equivalent to the one described in Section 1.3. of Part II to extract answer from a Davis's oracle by extracting the binary expansion of a real number.

These systems can perform computations as other abstract machines such as automata and Turing machines. In what follows, we aim to classify these nets by their computational power, discuss their complexity and, our final goal, present relativization results.

[10]But this is not true in general. For example, if the input is 01, in the first one the output will be 11 while in the second one it will be 111. In the radical case where the input is simply 0, in the first system the result will be the empty word while in the other it will be 11! We say that 0 is not *classified* by the first system. This definition will be given and explored in the next section.

In fact, if we consider $\{0, 1\}^+$ as the domain of the input line, the output of the first neural net is the following: if α_n, the n^{th} digit of the input, is 1, then $(\phi(\alpha))_n = (\phi(\alpha))_{n+1} = 1$; otherwise, it is 0. When two consecutive 0 is imported, the net will attain an *equilibrium*, that is, its state at that instant is $\vec{0}$. Comparing with the second net: $\phi(\alpha)$ is $|\alpha| + 1$ in unary.

[11]Note that if the summands are equal, there is nothing to be saved.

Chapter 3

Computational model

In the last chapter, a simple example of addition was exhibited. Can we implement simple logical predicates such as OR, AND, NOT? We will show that there are simple modules that can compute these functions and how they can be connected to build more complex modular constructions. As parallel computations are allowed, synchronization plays an important role. In the last part of this chapter, we will be directing to it, showing how to connect various subsystems.

The inputs are given in binary and the activation function of each processor here will be the saturated sigmoid

$$\sigma(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

3.1 Basic logical predicates

From Propositional Calculus we know that all quantifier-free propositions can be expressed with 0-ary predicates 0 and 1 (meaning False and True, respectively), unary predicate NOT and binary predicates AND and OR. The following neural nets can simulate these basic predicates.

The 0-ary predicates can be expressed by a single unit with the dynamics

$$z_0^+ = \sigma(0) \;\text{ and }\; z_1^+ = \sigma(1).$$

The unary function NOT is computed by

$$z_\neg^+ = \sigma(-u+1)$$

The predicates AND and OR given as input u_1 and u_2 can be simulated by

$$z_{AND}^+ = \sigma(u_1 + u_2 - 1) \qquad\qquad z_{OR}^+ = \sigma(u_1 + u_2)$$

By composing these units, we can compute any logical proposition. To be able to distinguish between an output unit in rest and sending the signal False, we use an extra unit called the validation of the output. When this unit holds the value 1, the value of the output unit holds the computation result. Similarly, we add an extra input validation line.

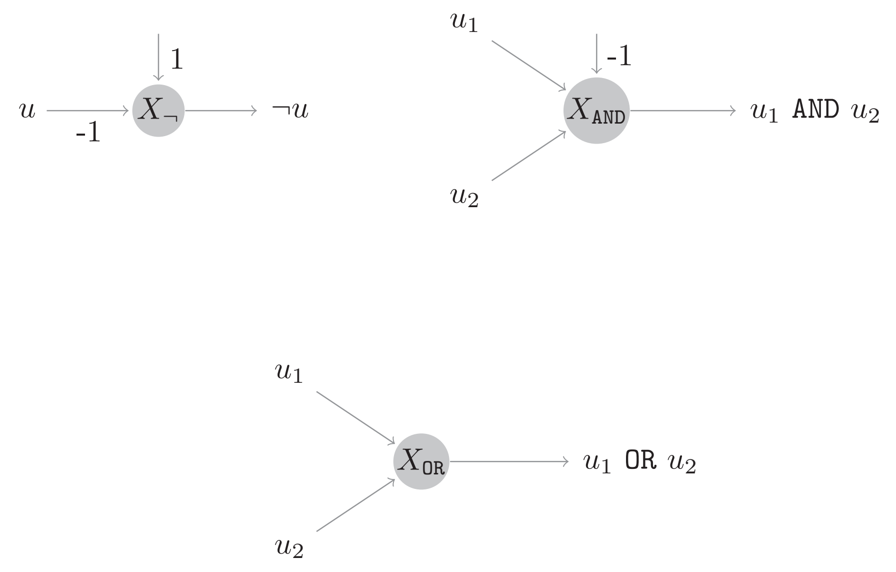


Figure 3.1: Neural nets that computes the logical predicates `NOT`, `AND` and `OR`. In this representation, circles denotes the units. The label on the arrows denotes the corresponding weights. When there is no label, the corresponding weight is 1.

Example Let $\varphi(u_1, u_2, u_3) = u_1 \texttt{ AND } (u_2 \texttt{ OR } \neg u_3)$. Clearly we will need three layers. The first one to compute $\neg u_3$ and to hold the values of u_1 and u_2

$$\begin{cases} x_{1,1}^+ = \sigma(u_1) \\ x_{1,2}^+ = \sigma(u_2) \\ x_{1,3}^+ = \sigma(-u_3 + 1) \end{cases}$$

A second layer to compute the disjunction and hold the value of u_1

$$\begin{cases} x_{2,1}^+ = \sigma(x_{1,1}) \\ x_{2,2}^+ = \sigma(x_{1,2} + x_{1,3}) \end{cases}$$

The last one is to compute the function φ. The only unit here is the output unit.

$$x_\varphi^+ = \sigma(x_{2,1} + x_{2,2} - 1)$$

In three steps, the solution is computed. So, denoting the input validation line by v, we add some extra units to compute the output validation d

$$\begin{cases} y_1^+ = \sigma(v) \\ y_2^+ = \sigma(y_1) \\ y_d^+ = \sigma(y_2) \end{cases}$$

Let us compute $\varphi(1, 0, 0) = 1$. This table shows a step by step simulation of this computation given input $u_1 = 1, u_2 = u_3 = 0$ at $t = 1$.

t	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,1}$	$x_{2,2}$	x_φ	y_1	y_2	y_d
1	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	1	0	0
3	0	0	1	1	1	0	0	1	0
4	0	0	1	0	1	1	0	0	1
5	0	0	1	0	1	0	0	0	0

3.2 Memory and local inhibition

As cycles in single processors are allowed, one can easily understand that a single unit have capacity to hold a value (forever if needed). Given input u, a neuron x following the dynamics

$$x^+ = \sigma(x + u)$$

can save the value of u. Once u feeds in some value, x will hold this value forever. To build a more complex unit x that saves the last value introduced, consider one input line u and an input validation line v.

$$x^+ = \sigma(x + 2u - v)$$

One can easily check that it works as explained by evaluating for every possible state and input.

x	u	v	x^+
0	0	0	0
0	0	1	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	1	1

Suppose now that we have another unit, y, and we want it to download the value of x when it receives as input 1 from input line u. This download unit can follow, for example, the dynamics

$$y^+ = \sigma(x + u - 1)$$

The input line u can be seen as a *switch*. This can be generalized. Suppose that a unit y is defined as

$$y^+ = \sigma\left(\pi\left(x_{i_1}, \ldots, x_{i_k}\right) + bu - b\right)$$

where $\pi : [0,1]^k \longrightarrow \mathbb{Z}$ is a linear function[1] with integer coefficients $a_1, \ldots, a_k$. Let a_l be the coefficient with greatest absolute value among them and $c = |a_l|$. Then the values held by π are within $[-ck, ck]$. By setting $b = c + 1$ we guarantee that:

1. if $u(t) = 0$, then $y(t+1) = \sigma\left(\pi\left(x_{i_1}(t), \ldots, x_{i_k}(t)\right) - b\right) = 0$;

2. if $u(t) = 1$, then $y(t+1) = \sigma\left(\pi\left(x_{i_1}(t), \ldots, x_{i_k}(t)\right)\right)$.

[1]For the constant term, consider $x_{i_k}(t) = 1, \forall t \in \mathbb{Z}$.

Lemma 3.2.1 *(Switch Lemma) Let y be a unit with dynamics*

$$y^+ = \sigma\left(\pi\left(\vec{x}\right)\right).$$

Then there exists a substitution of y by $\tilde{y}$ described as

$$\tilde{y}^+ = \sigma\left(\tilde{\pi}\left(\vec{x}, x_{switch}\right)\right)$$

such that if $x_{switch}(t) = 0$ then $\tilde{y}(t+1) = 0$ and if $x_{switch}(t) = 1$ then $\tilde{y}(t+1) = y(t+1)$.

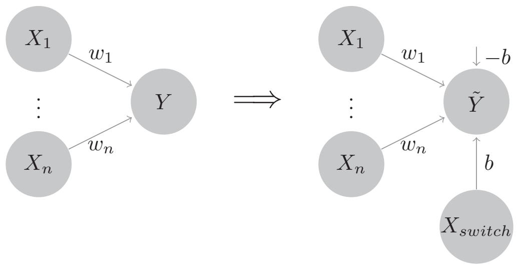


Figure 3.2: Switch Lemma

Remark This is generally true for neural networks with real weights and activation functions with bounded values. The proof is similar.

Remark By considering $y^+ = \sigma(\pi(x_{i_1}, \ldots, x_{i_k}) - bu)$ we can have the opposite combination.

3.3 Synchronization

Definition 3.3.1 *A **subsystem** is a (real) neural network where inputs and outputs allowed must not be binary.*

Note that a conventional neural network is a particular kind of subsystem but the reverse is not always true.

Let Σ_1 and Σ_2 be two subsystems, where Σ_1 has two output units x_{out} and x_{outv} (denoting output and output validation, respectively) and Σ_2 has two input lines u_{in} and u_{inv} (denoting input and input validation, respectively) connected to units $x_1, \ldots, x_n$.

To connect Σ_1 to Σ_2, add two extra units, x_{in} and x_{inv}, to Σ_2 such that

$$x_{in}^+ = \sigma(x_{out}) \qquad x_{inv}^+ = \sigma(x_{outv})$$

Then, simply change $x_1, \ldots, x_n$ by substituting in their dynamics u_{in} and u_{inv} by x_{in} and x_{inv} respectively.[2] After downloading, $x_{outv}(t) = 0$ and $x_{inv}(t+1) = 0$ but the input unit x_{in} still needs to be switched off at $t+1$. We can add a switch as in Lemma 3.2.1 to x_{in}, that is, change its dynamics to

$$x_{in}^{+} = \sigma(x_{out} + x_{outv} - 1)$$

Suppose that Σ_1 is in rest at $t = 0$ and that $x_{outv} = 1$ at $t = T$ and for $t < T$, $x_{outv} = 0$. At time $T+1$, Σ_2 starts as expected, downloading the values of x_{out} and starting its computations at $T+2$.

Remark If we want to shut Σ_1 down after downloading the values, we can add a switch unit x_{switch}. If

$$x_{outv}^{+} = \sigma\left(\pi\left(\vec{x}\right)\right)$$

let b be the constant in Lemma 3.2.1. Set

$$x_{switch}^{+} = \sigma\left(-\pi(\vec{x} + b(x_{outv} - 1) + bx_{switch}\right)$$

and add to the dynamic of every unit x_i in Σ_1 (including x_{outv}) an extra term $-b_i x_{switch}$, where b_i is the constant of Lemma 3.2.1 for the respective unit.

Proposition 3.3.2 *Let Σ_1 and Σ_2 be two subsystems. Then we can connect them to build a new subsystem working as follows. Σ_1 starts its computation while Σ_2 is in rest until the output of Σ_1 is fed into Σ_2. If needed, Σ_1 can be shut down after Σ_2 downloaded its output.*

We just showed that we can perform *combination of subsystems in sequence* by iterating the last proposition. Now, consider two subsystems Σ_1 and Σ_2 working in different time and we have a third subsystem Σ waiting to receive the outputs from output units x_{out}^1 and x_{out}^2 of Σ_1 and Σ_2. We want to feed their outputs at the same time into Σ. This is a synchronization problem in combining subsystems *in parallel*.

Without loss of generality, suppose Σ is composed by only one unit, x, with two input lines, u_1 and u_2, receiving signals from Σ_1 and Σ_2.

For Σ_i, add two units o_i and v_i, where

$$o_i^{+} = \sigma\left(o_i + x_{out}^i\right) \qquad v_i^{+} = \sigma\left(v_i + x_{outv}^i\right)$$

These units holds the outputs and its validation. A switch unit x_{switch} works as follows[3]

$$x_{switch}^{+} = \sigma\left(-x_{switch} + v_1 + v_2 - 1\right)$$

To the system Σ add two extra units x_{in}^1 and x_{in}^2 given by

$$x_{in}^i = \sigma\left(o_i\right)$$

and substitute u_i by x_{in}^i in Σ.

Now, to finish, alter v_i, o_i and x_{in}^i as in Lemma 3.2.1.

[2]The units x_{in} and x_{inv} are there because $x_1, \ldots, x_n$ can depend also on other units of Σ_2. We do not want to switch off units that make part of the computations of Σ_2. If these units only depends on u_{in} and u_{inv}, then we can connect them directly to x_{out} and x_{outv} of Σ_1.

[3]The constant term is $-(n-1)$ when we need to combine n systems. The term $-x_{switch}$ is a switch to turn itself off.

Proposition 3.3.3 *Let Σ_1 and Σ_2 be two subsystems with one output unit and Σ with two input lines. Then we can build a subsystem that works as follows. Σ_1 and Σ_2 start computing at $t = 0$ while Σ is in rest. The output of the first subsystem finishing the computation is saved until the second finishes its computation. When both finish their computations, the outputs are fed into Σ, starting its computation.*

Remark We can turn off the subsystems being synchronized after the latter finishes downloading the signals as in the sequential case.

This is just an unfolded corner of a whole page of discussion about the programmability of neural nets and logical description of realizable logical propositions. For further results, see [10], [11] and [14]. We will move on to the characterization of the languages accepted by these nets as a computational model.

Chapter 4

Characterization of computational model

As we have a new abstract device, a natural question arises. How are these neural nets related to existing machines? Is there any uniform classification of languages using neural nets? In Appendix A, finite automata and Turing machines are defined. Readers with basic computational background can skip it and direct towards the word classification problem in ARNN.

4.1 Word classification in ARNN

Given a word built with an alphabet, we should now look for systems that recognizes it. More generally, one should ask whether a system can recognize a language. First, let us give a formal definition of what we mean by recognition:

Definition 4.1.1 *A word $\alpha \in \{0,1\}^+$ is said to be **classified in time** ν by a system Σ if the input streams are $\langle A, V\rangle$, with $A = 0\alpha 0^\omega$ and $V = 01^{|\alpha|}0^\omega$ and the output streams are $\langle U, R\rangle$ with $R(t) \equiv (t = \nu)$. If $U(\nu) = 1$, then the word is said to be **accepted**, otherwise (if $U(\nu) = 0$) **rejected**.*

In the following sections, we will discuss the computational power of systems such as rational ARNN and, in Part II, real ARNN. The classes of languages decided by systems in integral, rational and real ARNN will be denoted respectively by $ARNN[\mathbb{Z}]$, $ARNN[\mathbb{Q}]$ and $ARNN[\mathbb{R}]$.

4.2 Integral ARNN

Neural nets with integer weights are those introduced by McCulloch and Pitts [11]. Since the state variables only hold linear combinations of 0 and 1, the obvious activation function is the McCulloch-Pitts sigmoid

$$\sigma(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

McCulloch and Pitts [11] and Kleene showed how to simulate finite automata by integral nets. Here we will adopt the prove in the book of Minsky [12].

Definition 4.2.1 *A language is said to be* ***regular*** *if there is a finite automaton deciding its words.*

Theorem 4.2.2 *L is a regular language if and only if $L \in ARNN[\mathbb{Z}]$.*

Proof This theorem can be proved by providing how to simulate a neural net by a finite automaton and vice versa.

Let Σ be an integral net of dimension n with m input lines[1]. Note that the n neurons of Σ hold binary values totalling 2^n possible states.

Build an automaton M with 2^n states and using as alphabet binary vectors of dimension m, corresponding to all possible states of Σ and all the possible inputs. Define the transition function by the computing laws of each state of Σ.

Consider the states of M that represent the states of Σ where the output and output validation units are activated. By setting them as final states of M, this finite automaton will accept the same language as Σ.

We shall prove the converse. Let $M = \langle Q, \Sigma, \delta, q_0, \mathcal{F} \rangle$ be a finite automaton with p states $q_1, \ldots, q_p$ and alphabet $\{s_1, \ldots, s_m\}$. Set a neural net Σ_M with m input lines $x_1, \ldots, x_m$, each corresponding to one letter. So in each instant there is only one line fed with value 1.

Consider units u_{ij}, $0 \leq i \leq p$, $1 \leq j \leq m$, and an extra unit u_0. The dynamics of this system will be defined as follows.

At $t = 0$, unit u_0 holds value 1 and the others 0. Fix $q_i \in Q$ and $s_j \in \Sigma$. Consider $q_{i_\ell} \in Q$ such that $\delta(q_{i_\ell}, s_j) = q_i$. Then, the dynamic map of the unit u_{ij} will be:

$$u_{ij}^{+} = \sigma \left(\sum_{\ell} \sum_{k=1}^{m} u_{i_\ell k} + s_j - 1 \right).$$

Note that in each instant after $t = 0$, only one unit is active in the system, which is equivalent to the definition of the transition map δ. That is, u_{ij} fires at time $t+1$ if and only if at time t one of the units $u_{i_1 1}, \ldots, u_{i_1 m}, \ldots, u_{i_\ell 1}, \ldots, u_{i_\ell m}$ (and possibly u_0) and the unit s_j fired. This mimics the transitions of M in the sense that if in instant t the automaton is in state q_i, then the only active unit of Σ_M at time t will be among $u_{i1}, \ldots, u_{im}$.

By implementing one extra unit u_{out} that sums up all the units corresponding to the final states of M, we will obtain a neural net of dimension $m \times p + 1$ that simulates M. ▮

We just proved that integral nets and finite automaton have the same power. Is there any analogy to rational or real nets? It turns out that rational nets are equipotent to Turing machines, proved by Siegelmann and Sontag [17]. As we will see, the fact that rational numbers can hold as many digits (but just finitely many!) as we want is the key to the simulation of a Turing machine, an abstract device with infinite resource writing only finite information in each step.

[1] Although we only need two input lines, we prefer to leave this part of the proof as general as possible.

4.3 Rational ARNN

In rational neural nets, the use of McCulloch-Pitts sigmoid makes no sense anymore. Instead, the saturated sigmoid will be applied as activation function

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

That is, if $x \in [0,1]$, then $\sigma(x) = x$. This will allow us to encode information of arbitrarily large size. For instance, consider words α in $\{0,1\}^+$. Using $[\cdot]_4$ to encode these streams of values

$$[\alpha]_4 = \sum_{i=1}^{|\alpha|} \frac{2\alpha_i + 1}{4^i}$$

$\{0,1\}^+$ is mapped to a subset $\mathcal{C}$ of the rational numbers (actually, the image of this function is a Cantor subset of the rationals; for further details, see Appendix B). The following pair of functions will retrieve bit by bit the encoded word

$$\mathit{top}(q) = \sigma(4q - 2) \qquad \mathit{pop}(q) = \sigma(4q - (2\mathit{top}(q) + 1))$$

Together with the functions that concatenates 0 or 1 on the left of a word

$$\mathit{push}0(q) = \sigma\left(\frac{q}{4} + \frac{1}{4}\right) \qquad \mathit{push}1(q) = \sigma\left(\frac{q}{4} + \frac{3}{4}\right)$$

and one that evaluates if a word is empty, i.e., giving 0 if the word is λ and 1 otherwise,

$$\mathit{nonempty}(q) = \sigma(4q),$$

we have the basic functions that "encode" the dynamics of a many taped Turing machine to a system over the rationals in $[0,1]$.

As part of the definition, Turing machines have finite control. Denote it by Q with s states. Then, we can describe it unarily by the cannonical vectors $e_i = (0, \ldots, 1, \ldots, 0)$ in $\mathbb{Q}^s$, with all entries 0 except for the ith coordinate with value 1.

To describe the transitions from one state to another of a p-taped Turing machine, we have the *transition map*, denoted by δ, given in its defintion. This can be extended to its *complete dynamic map* $\mathcal{F} : Q \times \{0,1\}^p \longrightarrow Q \times \{0,1\}^p$, describing every transition.[2]

Encoding through $\pi : Q \times \{0,1\}^p \longrightarrow \mathbb{Q}^{s+p}$ all the descriptions of the machine and through the "encoding" of the tape operations, we get a new map $\tilde{\mathcal{F}} : \mathbb{Q}^{s+p} \longrightarrow \mathbb{Q}^{s+p}$ making the following diagram commutative

$$\begin{tikzcd} Q \times \{0,1\}^p \arrow[r, "\pi"] \arrow[d, "\mathcal{F}"] & \mathbb{Q}^{s+p} \arrow[d, "\tilde{\mathcal{F}}"] \\ Q \times \{0,1\}^p \arrow[r, "\pi"] & \mathbb{Q}^{s+p} \end{tikzcd}$$

[2]This map holds the words in every tape of the machine instead of the position of the head.

By composing $\tilde{\mathcal{F}}$ into four transition functions

$$\tilde{\mathcal{F}} = F_1 \circ F_2 \circ F_3 \circ F_4$$

each F_i simulable by a subsystem, we get a four-layered neural net.

In fact, this is a sketch of the essential part of the proof of

Theorem 4.3.1 *If $L \in \{0,1\}^+$ is decidable (in the sense of Turing) in time t, then there exists a rational system Σ such that, for every word $\alpha \in \{0,1\}^+$, the system classifies α in time $O(t(|\alpha|) + |\alpha|)$.*

In this simulation, each step of the Turing machines is mimicked in four steps. Changing the encodings of the tapes, $\alpha \in \{0,1\}^+$ to

$$\sum_{i=1}^{|\alpha|} \frac{10p^2 - 1 + 4p(\alpha_i - 1)}{(10p^2)^i}$$

these systems can simulate the respective Turing machines in real time. For complete proof, see [17].

We refer to this as *real time simulation of Turing machines by neural nets*. This result is crucial for many proofs of our work here.

The simulation of neural nets by Turing machines is trivial. If a neural net accepts a word α in time $t(\alpha)$, then there is a Turing machine accepting α in time $O(p(t(\alpha)))$.[3]

Definition 4.3.2 *A language (or set) $L \subseteq \{0,1\}^+$ is said to be* ***recursively enumerable*** *if there exists a system Σ such that, for every word α, (a) if $\alpha \in L$, then α is accepted by Σ, i.e., it is classified at some time and accepted, and (b) if $\alpha \notin L$, then either α is never classified or it is classified and rejected.*

Definition 4.3.3 *A language (or set) $L \subseteq \{0,1\}^+$ is said to be* ***recursive*** *if there exists a system Σ such that, for every word α, if $\alpha \in L$ then α is accepted by Σ, else (if $\alpha \notin L$), then α is rejected by Σ.*

These definitions are sound, i.e., in agreement with the definition of the concepts with the same name in classical theory.

Theorem 4.3.4 *A language L is recognizable if and only if $L \in ARNN[\mathbb{Q}]$.*

Since the simulation of a Turing machine is done in real time and the other way around in polynomial time, the polynomial and exponential classes of recognizable languages is preserved. That is, denoting by $ARNN^P[\mathbb{Q}]$ and $ARNN^E[\mathbb{Q}]$ classes of languages accepted by rational nets in polynomial and exponential time,

Corollary 4.3.5 *$P = ARNN^P[\mathbb{Q}]$ and $EXPTIME = ARNN^E[\mathbb{Q}]$.*

Until now, we have been analysing deterministic neural net models. In the next section, we will define a non-deterministic version of these processor nets and explore their properties. We recall that our final goal is to exhibit relativization results. Hence, the relation between deterministic and non-deterministic systems should be clarified.

[3]Consider tapes that keeps the value of each unit of the net being simulated. Multiplication, sum and evaluation by the activation function can be performed in polynomial time. Note that the weights are fixed rationals.

4.4 Non-determinism in ARNN[$\mathbb{Q}$]

Here, we will consider nets with rational weights and saturated sigmoid.

First, a definition of non-deterministic processor net:

Definition 4.4.1 *A **non-deterministic analog recurrent neural net** (NDNN) $\mathcal{N}$ (of dimension m) consists on three input units of **validation of inputs**, **input** and a **guess unit**, receiving streams $v = 01^{|\alpha|}0^{\omega}$, $u = 0\alpha0^{\omega}$ and γ, a guess stream, with dynamics defined by*

$$\vec{x}(t+1) = \sigma(A\vec{x}(t) + \vec{b_1}v(t) + \vec{b_2}u(t) + \vec{b_3}\gamma(t) + \vec{c})$$

where $\vec{x}$ is the state vector of dimension m, A an $m \times m$ matrix, $\vec{b_i}$ and $\vec{c}$ vectors of dimension m.

*Two special units are chosen for the **output validation** and the **output**, sending out streams $z = 0^{T_{\mathcal{N}}(\alpha)-1}1^{|\tilde{\phi}(\alpha)|}0^{\omega}$, $y(t) = 0$ for $t < T_{\mathcal{N}}(\alpha)$ and $y(T_{\mathcal{N}}(\alpha) - 1 + i) = (\tilde{\phi}(\alpha))_i$, where $T_{\mathcal{N}}$ is the **computation time** given α as input and $\tilde{\phi} : \mathbb{N} \longrightarrow \mathbb{N}$ the function computed by $\mathcal{N}$. A word is in dom($\tilde{\phi}$) if there is a guess stream such that its computation will lead to an output validation $z(t) = 1$ for some t.*

Note that if we impose a time bound t, then only the first $t(\alpha)$ digits of the guess stream is needed.[4] We can regard the guess stream (which only admits binary values as other input streams) as a path in the binary tree for possible sets of states of the given net. Each branch corresponds to a choice in $\{0, 1\}$. The values that the G unit takes decide the path of sets of states as a guess in a non-deterministic Turing machine does with its transition map (see Appendix A). We will explore this in the next section.

In general, the function $\tilde{\phi}$ computed by a NDNN receives as argument α and γ. If a word α is in its domain, then its value can vary for different streams γ that lead to acceptance, so that $\tilde{\phi}$ is multi-valued: a partial funcion $\Phi : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ can be defined, where $\Phi(\alpha, \gamma_1)$ takes value $\tilde{\phi}(\alpha)$ given γ as guess and γ_1 one of its prefix. We are interested in a more restricted definition of NDNN, those that computes functions $\phi : \mathbb{N} \longrightarrow \mathbb{N}$, that is, given $\alpha \in \{0, 1\}^+$, there is a γ such that $\Phi(\alpha, \gamma_1)$ is defined and, for all such γ this value should be the same. In this case, we write $\phi(\alpha) = \Phi(\alpha, \gamma_1)$. Compare this restricted definition to the following:

Definition 4.4.2 *A function ϕ is in NPF if it is computed in polynomial time by a non-deterministic Turing machine M, that is:*

(a) M accepts the domain of ϕ

(b) if $\langle x_1, \ldots, x_n\rangle \in$ dom(ϕ), then any accepting computation writes in the output tape the value $\phi(x_1, \ldots, x_n)$ in polynomial time

The notion of acceptance of a language is similar to the one for non-deterministic Turing Machine:

[4]Synchronization is used to clock the computation.

Definition 4.4.3 *A language L is said to be* ***decided*** *by a non-deterministic net $\mathcal{N}$ if for all $\alpha \in L$, there is a guess γ such that $\Phi(\alpha, \gamma_1) = 1$, that is, $\mathcal{N}$ computes its characteristic function*[5].

This particular case coincides with the definition of a NDNN given by Siegelmann and Sontag.

4.5 Polynomial time bound

In [17], it is stated that one can obtain a simulation of non-deterministic Turing machines by NDNN in a similar way to the deterministic case. Let us assume in this section the special case where all the computation held by NDNN can be done in polynomial time. It seems intuitive that these nets are equivalent to non-determinitic Turing machines polynomially bounded. Here we aim to show a clear proof of this statement.

Proposition 4.5.1 *If ϕ is a funcion computed by a NDNN bounded in time by a polynomial, denoted $\phi \in ARNN^{ND}PF$, then $\phi \in NPF$.*

Proof Let ϕ be a function of $ARNN^{ND}PF$, computed by a non-deterministic net $\mathcal{N}$, working in time $T_{\mathcal{N}}$ bounded by p a polynomial, as in Definition 4.4.1.

Build a deterministic Turing machine M which simulates $\mathcal{N}$ given an input α and a guess γ.[6]

Now, build the following non-deterministic Turing machine:

```
procedure:
        begin
              input $\alpha$;
              guess $\gamma$ such that $|\gamma| \leq p(|\alpha|)$;
              simulate $M$ on $\langle \alpha, \gamma \rangle$ clocked by $p$;
              if $M$ is in accepting state, output its result
        end
```

This machine witnesses the fact that $\phi \in NPF$, since addition and multiplication can be done in polynomial time and polynomials are closed under composition. ∎

Once proven the other inclusion, we will have the equivalence

Theorem 4.5.2 $ARNN^{ND}PF = NPF$

The only tool we have is simulation of deterministic Turing machines. The following definition and proposition provide an alternative definition of NPF, which consists on separating a general non-deterministic Turing machine in a guess part attached to a deterministic Turing machine.

Definition 4.5.3 *The class $\exists PF$ consists on functions $\phi : \mathbb{N}^n \longrightarrow \mathbb{N}$ such that there exist a function $\Phi : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ in PF and a polynomial p with the properties:*

[5]The characteristic function of a set A is given by $\chi_A(\alpha) = 1$ if $\alpha \in A$, else $\chi_A(\alpha) = \perp$ (undefined).

[6]Note that $\mathcal{N}$ can be regarded as deterministic once $\langle \alpha, \gamma \rangle$ is given as input.

(a) $\langle x_1, \ldots, x_n\rangle \in dom(\phi)$ *if and only if there exists* k *such that* $|k| \leq p\left(\sum_{i=1}^{n} |x_i|\right)$ *and* $\langle x_1, \ldots, x_n, k\rangle \in dom(\Phi)$*;*

(b) $\phi(x_1, \ldots, x_n)$ *is defined and its value is* y *if and only if there exists a* k *such that* $|k| \leq p\left(\sum_{i=1}^{n} |x_i|\right)$*,* $\langle x_1, \ldots, x_n, k\rangle \in dom(\Phi)$*,* $\Phi(x_1, \ldots, x_n, k)$ *is defined and, for all* k *in these conditions,* $\Phi(x_1, \ldots, x_n, k) = y$*.*

Proposition 4.5.4 $NPF = \exists PF$

Proof Let $\phi \in NPF$ witnessed by the non-deterministic Turing machine M. The following machine computes the function Φ with the properties of Definition 4.5.3:

```
procedure:
        begin
                input $x$ and $z$;
                simulate $M$ on $x$ using $z$ as guess;
                if $M$ is led to the acceptance state
                then output the value of the output tape
                else reject
        end
```

Conversely, let $\phi \in \exists PF$ and the correspondent $\Phi \in PF$ witnessed by Turing machine M and polynomial p. We can construct a non-deterministic Turing Machine which computes ϕ in polynomial time:

```
procedure:
        begin
                input $x$;
                guess $z$ such that $|z| \leq p(|x|)$;
                if $M$ is led to the acceptance state
                then output the value of the output tape
        end
```

This ends the proof. ∎

As a corollary, we have that $NP = \exists P$ by the last results applied to characteristic functions. This provides a proof for a different characterization of NP: A is in NP if and only if there is a set B in P and a polynomial p such that $x \in A$ if and only if $\exists z, |z| \leq p(|x|) : (\langle x, z\rangle \in B)$.

Now, we are ready to prove Theorem 4.5.2.

Proof Suppose $\phi \in NPF = \exists PF$. Let M be the Turing machine that computes the correspondent Φ. Note that M is a deterministic Turing machine with two input tapes and its computation is done in polynomial time. We can then simulate it in real time by a deterministic neural net $\mathcal{N}$ with one validation input line and two input lines, one of them for the guess stream. ∎

Hence, the class of languages accepted by these nets is exactly NP. Joining this with the fact that deterministic neural nets with polynomial time bound decides exactly P, one is induced naturally to the positive relativization of the Hypothesis $P = NP$ discussed in Part II.

Part II

Relativization in ARNN

Chapter 1

Oracles

In this chapter, we will recall the basic definitions of oracle Turing machine and advice (so the first section can be skipped by readers familiar with them). By inserting an extra real weight in rational ARNN, one have an effect similar to that when an oracle is used in a Turing machine. We will introduce the notion of Davis's oracle which will allow us to simulate such ARNN by an oracle Turing machine and vice versa. In the next chapters, we will move towards relativization results, which is the final goal of this work.

1.1 Oracles in Standard Computation

In 1938, Alan Turing described in his doctoral thesis a new kind of machine, what he called the *O-machine*, which has been considered to be a way to achieve the so-called "uncomputable" or "hypercomputation" by computer scientists such as Copeland and Proudfoot [6].

Definition 1.1.1 *An* ***oracle Turing machine*** *M is a Turing machine with a special tape called the* ***query tape****, three special states q_{query}, q_{yes} and q_{no} and O, a set called the* ***oracle*** *set, following the conditions:*

1. *when the machine is in the query state q_{query}, the machine stops its computation and in one computation step verifies if the word in the query tape, say w, is in O;*

2. *if $w \in O$, then M transits to q_{yes};*

3. *if $w \notin O$, then M transits to q_{no};*

4. *after the oracle's answer, M continues its computation.*

The main idea of this oracle Turing machine is to enrich a standard Turing machine with a blackbox. This blackbox answers the membership question, that is, decides a given word, in one step. Note that the oracle can be an arbitrary set: computable, not computable... If one enhance a Turing machine with a noncomputable set as oracle, its noncomputability suggests that this new machine can decide more sets. Consequently, a natural computability hierarchy arises. *Can we build a Turing machine that accepts A using set B as an oracle?*

Definition 1.1.2 *A set A is **Turing-reducible** to a set B, written $A \leq_T B$, if A can be decided by a Turing machine with oracle B.*

The relation of Turing reducibility is in fact a preorder. Two sets X, Y are *Turing-equivalent*, written $X \equiv_T Y$ if $X \leq_T Y$ and $Y \leq_T X$. The equivalence classes generates a hierarchy called *Turing degrees.*

In Complexity Theory's point of view, an oracle may be seen as a computation accelerator. We can obtain what is called *the polynomial hierarchy* by iterating the following definition:

Definition 1.1.3 *A set A is **polynomial-time Turing-reducible** to a set B, written $A \leq_T^p B$, if A can be decided by a Turing machine with oracle B in polynomial time.*

We will also refer to the concept of advice.

Definition 1.1.4 *Let $\mathcal{A}$ be a class of sets and $\mathcal{F}$ a class of total functions $\mathbb{N} \longrightarrow \Sigma^*$. The **non-uniform class** $\mathcal{A}/\mathcal{F}$ is the class of sets B such that there exist $A \in \mathcal{A}$ and $f \in \mathcal{F}$ such that*

$$x \in B \textit{ if and only if } \langle x, f(|x|)\rangle \in A.$$

*f is said to be an **advice function**.*

Example *P/poly* is the class of sets decidable by deterministic Turing machines working in polynomial time with advice of polynomial size. Note that f may not be computable. A similar example is *P/log*.

Example *P/exp* has an analogous definition. This class is in fact $\mathcal{P}(\{\prime, \infty\}^+)$. Given a set A, we can build a word of length 2^n for each n, called the characteristic of the set A, such that in lexicographical order the ith word with size n is in A if and only if the ith digit of the advice is 1.

1.2 Oracles in Non-Standard Computation – Hypercomputation

Families of circuits of polynomial size have been known to decide problems in *P/poly* (see [2]). In the article [16], Siegelmann and Sontag proved that one can simulate them by a rational neural net with one real weight.

Definition 1.2.1 *A **circuit** is a directed acyclic graph, where nodes of in-degree 0 are called **input nodes** and the others **gates** labelled by one of the Boolean functions AND, OR, or NOT, computing the correspondent function. The first two types are of many variables and the third a unary function. A special node with no outgoing edge is designated as the **output** node. The **size** of a circuit is the total number of gates. Arrange a circuit by **levels** $0, \ldots, d$ so that the input nodes are in level 0, the output nodes in level d and each level have gates only with ingoing egdes from gates of the previous level. The **depth** is then d. A **family of circuits** is a set of circuits*

$$\{c_n : n \in \mathbb{N}\}.$$

Theorem 1.2.2 *(a) There is an injective enconding from the set of families of circuits to the 9-Cantor subset of* $[0,1]$*; (b) if r encodes a family* $(A_k)_{k\in\mathbb{N}}$ *of polynomial size circuits, then the code of the nth circuit can be found among the first* $p(n)$ *digits of the decimal expansion of r, where p is a polynomial depending on* $(A_k)_{k\in\mathbb{N}}$*, and furthermore (c) we can construct an ARNN – call it* N_r *– with weights in* $\mathbb{Q}\cup\{r\}$ *to extract, given input z of size n, the code of* A_n *in a number of steps bounded by* $p(n)$*.*

The proof of this result relies on a subsystem called *BAM* which recovers bit by bit such encodings (see Section 1.3 for proof). This shows that the class of languages decided by rational neural nets working in polynomial time with one real weight is at least as powerful as *P/poly*.

Proposition 1.2.3 *P/poly* $\subseteq ARNN^P[\mathbb{R}]$*, where* $ARNN^P[\mathbb{R}]$ *denotes the class of sets decided by neural nets with real weight working in polynomial time.*

Proof Suppose A is a set in *P/poly* and let x be a word of size n. By the last theorem, we can build N_r which extracts the code of A_n. Feeding $\langle x, A_n\rangle$ into $NCVP$, a rational neural net that simulates CVP in real time, we can decide in polynomial time if $x \in A$. ∎

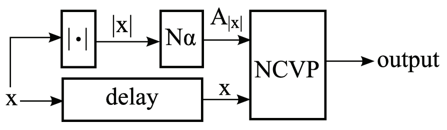


Figure 1.1: A neural net that simulates a family of circuits.

In fact, neural networks working in polynomial time with real weights decide exactly the sets in *P/poly*. Moreover, real neural nets working in exponential time can decide any set! And here arises the claim of hypercomputational power of recurrent neural nets made by Siegelmann. *If we can implement such a real weight in a neural net*, a computer with hypercomputational properties can be built, exceeding the Turing machines' computational power! Others proposed the use of physical constructs as oracles to access in a natural way real numbers encoded in the Universe.

Many defended that this is not possible by the impossibility of adjustment of the physical devices with arbitrary approximation. This misconception can be refuted by the following result:

Proposition 1.2.4 *The output of an ARNN after t steps is only affected by the first* $O(t)$ *digits in the expansion of the weights and the states in each computation time. More preciscely,* $Trunc_t = \left\lceil log\left(\frac{1}{8}(LW)^{(t-1)}\right)\right\rceil$ *digits of precision will suffice, where L is one plus the dimension of the net and the number of input lines, W is one plus the largest absolute value of its weights.*

From the proof of this proposition ([16]), it was showed that one can simulate real ARNN working in time t by a circuit of size $O(t^3)$. As a corollary, we can fully classify $ARNN^P[\mathbb{R}]$ and $ARNN[\mathbb{R}]$.

Theorem 1.2.5 $ARNN^P[\mathbb{R}] = P/poly$.

Theorem 1.2.6 *For any language there is a neural network with real weights which decides it. Conversely, an exponential time restriction is sufficient for real nets to decide any set. In resume,*

$$ARNN^E[\mathbb{R}] = ARNN[\mathbb{R}] = \mathcal{P}(\{0,1\}^+)$$

The last one results from the fact that, to decide any language, families of circuits of exponential size is enough. This full and uniform classification of languages decided by ARNN can be resumed as in Table 1.1.

Set of weights	Time restriction	Computational power
$\mathbb{Z}$	none	Regular languages
$\mathbb{Q}$	none	Recursive languages
$\mathbb{Q}$	t	DTIME(t)
$\mathbb{R}$	polynomial	$P/poly$
$\mathbb{R}$	none	All languages

Table 1.1: Computational power of ARNN under various restrictions.

In [7], Martin Davis critized Siegelmann's point of view and also other attempts to the survival of hypercomputation claims:

Since the non-computability that Siegelmann gets from her neural nets is nothing more than the non-computability she has built into them, it is difficult to see in what sense she can claim to have gone "beyond the Turing limit".

As real numbers can "boost up" the computational power of rational ARNN, we will present real numbers in the remaining of this work as oracles and show that they are a restricted class of oracles. In memory of this criticism, we will refer them as *Davis's oracles*.

Remark To simulate neural net with real weights, one real weight suffices. This results from Theorems 1.2.2 and 1.2.5. This will imply that a finite number of Davis's oracles can be reduced to one, although a direct construction is not known. Therefore, it is enough to prove relativization results in ARNN for nets with only one Davis's oracle, that is, only one real weight.

1.3 BAM and the Prefix Retrieval Process

In linear time a prefix of r of length n can be extracted by running the following procedure:[1]

[1]If $|\Gamma| = k$, suppose without loss of generality that $\Gamma = \{0, 1, \ldots, k-1\}$.

```
procedure:
          begin
                input $n$;
                $\tilde{r} := \lambda$;
                for $i := 1$ to $n$ do
                $k := 0$
                    for $j := 1$ to $|\Gamma| - 1$ do
                        if $j < r_i$, then $k = k + 1$
                    end for;
                    $\tilde{r} := \tilde{r}k$;
                end for;
                output $\tilde{r}$
          end
```

In [16], Siegelmann and Sontag presented for the first time a subsystem that simulates this procedure to retrieve the prefix of a given weight which has encoded in it a family of circuits using 9-Cantor encoding.

Remark Note that we cannot simulate this procedure using real time simulation of Turing machines by neural nets. This procedure is not a Turing machine since the real number r is already implemented into the procedure and it is an infinite string.

Let $\mathcal{C}_b$ be the b-Cantor subset in $[0,1]$ and $r \in \mathcal{C}_b$ a real number with digits within $\{0, 2, \ldots, b-l\}$, where l is 1 if b is odd and 2 if b is even.[2] To extract its digits, we can first compare r with $k \in \{0, 1, \ldots, b-1\}$ through the family of functions[3]

$$\Lambda_k(r) = \sigma(br - k)$$

and then shift the encoding of r one bit to the left:

$$\Xi(r) = \sigma\left(\sum_{k}^{b-1} (-1)^k \Lambda_k(r)\right).$$

Ξ is the shift map. When the word is not trivial, for atmost one k even, the value of $\Lambda_k(r)$ is in $(0,1)$. If such k exists, for $0 \leq j < k$ we have $\Lambda_j(r) = 1$ and for $j > k$, $\Lambda_j(r) = 0$. The following map recovers the prefix of r along the extraction and saves it in the reverse order as $\tilde{r}$:

$$\Psi(r, \tilde{r}) = \sigma\left(\frac{\tilde{r}}{b} + \frac{2}{b}\sum_{j} \Lambda_{2j}(r)\right).$$

This dynamical system can be simulated by a four-layered fully wired net called *Bidirectional Associative Memory neural network*, or simply *BAM*:

$$\begin{cases} y^+ = \sigma\left(V + \sum_k^{b-1} (-1)^k x_k\right) \\ x_k^+ = \sigma(by - k), k \in \{0, \ldots, b-1\} \\ z_1^+ = \sigma\left(\frac{1}{b} z_2 + \sum_j \frac{2}{b} x_{2j}\right) \\ z_2^+ = \sigma(z_1) \end{cases} \tag{1.1}$$

[2] A "good" encoding is one isomorphic to a b-Cantor encoding. See Appendix B.

[3] In fact, for b even, verifying this for $k \in \{0, 1, \ldots, b-2\}$ is enough. For homogeneity of notation, we can include $b-1$ since $\Lambda_{b-1}(r) = 0$ for all r.

where V is the unit that uploads the encoding of r to the BAM. Once V sends in r, the units x_k extracts one digit of r at a time while z_1 keeps $\tilde{r}$.[4] Joining this subsystem to a clock (and some work of synchronization), we can control the number of digits to be extracted. By choosing 9-Cantor encoding, we have the BAM exhibited by Siegelmann and Sontag. This let us proof Theorem 1.2.2.

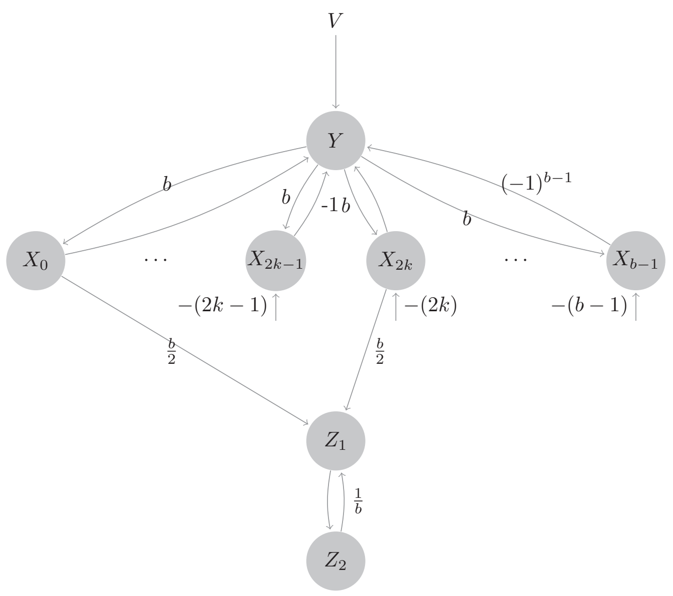


Figure 1.2: Architecture of BAM

Proof (a) Let A_k be one of the circuits from the family $C = (A_k)_{k\in\mathbb{N}}$. Enumerate the gates by layer and fix an order, say g_{ij} is the jth gate of level i. The encoding of the A_k, denoted by $[A_k]$, is performed as follows.

The enconding of a level starts with 6. Each level is encoded successively from the bottom level to the top one. In each level i, the gates are encoded by order, each g_{ij} starting with 0, followed by the respective code of the gate's label

$$\begin{aligned} AND &\longmapsto 42 \\ OR &\longmapsto 44 \\ NOT &\longmapsto 22 \end{aligned}$$

and then, by order, 4 if a gate $g_{i-1,j'}$ from the last level feeds into g_{ij} and 2 otherwise.

Denote by $\overline{w}$ the reverse of w. The encoding $[C]$ of C is given by

$$[C] = 8\ \overline{[A_1]}\ 8\ \overline{[A_2]}\ 8\ \overline{[A_3]} \ldots$$

The real number that represents $[C]$

$$r = \sum_{i\geq 1} \frac{[C]_i}{9^i}$$

[4]Unit z_2 is only for synchorization purpose.

encodes C in $\mathcal{C}_9$.

(b) Follows from the fact that $(A_k)_{k\in\mathbb{N}}$ is of polynomial size.

(c) The validation line of the input v feeds into the system the size of the input in unary. The unit

$$x^+_{|z|} = \sigma\left(\frac{1}{2}x_{|z|} + \frac{3}{2}v - 1\right)$$

encodes $|z|$ into a rational number $q_{|z|}$. Save this value into the unit x_q

$$\begin{cases} x^+_{delay} = \sigma(x_{|z|}) \\ x^+_q = \sigma(x_{delay} - v) \end{cases}$$

Build Σ, a BAM system for $\mathcal{C}_9$ extracting r. Each time Σ encounters an 8 in r, subtracts 1 from $|z|$, that is,

$$2q_{|z|} - 1$$

When $q_{|z|} = 0$, extract the bits of r until the next 8. Σ is given by

$$\begin{cases} x^+_i = \sigma(9x_{10} - i),\ i = 0, \ldots, 8 \\ x^+_9 = \sigma(2x_q) \\ x^+_{10} = \sigma\left(rx_9 + \sum_{i=0}^{8}(-1)^i x_i\right) \\ x^+_{11} = \sigma\left(\frac{1}{9}x_{12} + \frac{2}{9}(x_1 + x_3 + x_5 + x_7) - 2x_{13}\right) \\ x^+_{12} = \sigma(x_{11}) \\ x^+_{13} = \sigma(x_q + x_{14} + x_{15}) \\ x^+_{14} = \sigma(2x_{13} + x_7 - 2) \\ x^+_{15} = \sigma(x_{13} - x_7) \\ x^+_{16} = \sigma(x_{12} + x_7 - 1) \end{cases}$$

The units $x_0, \ldots, x_{12}$ forms the BAM. The unit x_{11} keeps the expansion of r. The control of this retrieval is done by the units $x_{13}, \ldots, x_{16}$ (note that in the dynamic map of x_{14}, $2x_{13} - 1$ counts $|z|$ downwards each time $x_7 = 1$ and x_7 works as a switch). ∎

The neuron Y in Eq. 1.1 can be regarded as the *query unit* and X_i *answer unit*. The time of extraction is linear on the numbers of digits required since each bit can be obtained in constant time. We say that this "oracle" is of *linear access time*. In the next sections, we will show how to simulate a rational net with one real weight by a Turing machine with a special type of oracle and then define access time protocols for oracles.

1.4 Davis's Oracles

The internalization of real numbers in machines or in neural nets has been seen as a path to achieve "non-computability". In this subection we provide a way of regarding real numbers as oracles.

As stated in Theorem 1.2.4, only the first digits of the expansion of the weights are needed. Consider the particular case of ARNN with real weights. The same output can be computed by substituting its weights by their first $O(t)$ bits if the computation is done in time t. Conversely, computations held by an ARNN with a real weight can atmost decide sets computable with its prefixes. This gives us the intuition that the oracles implemented in ARNN are a restricted class of oracles.

Definition 1.4.1 *A* ***Davis's oracle*** O_r *is a subset of* $Prefix(r)$ *for some* $r \in \Gamma^\omega$*, where* Γ *is the alphabet used by the working tapes.*[5] *The first* n *digits of* r *will be denoted* $r_{|n}$*.*

We will also call Davis's oracle sets of the kind

$$O_r^f = \{\langle 0^n, \tilde{r}\rangle : n \in \mathbb{N}, \tilde{r} \text{ is a prefix of } r_{|f(n)}\}$$

for some $f : \mathbb{N} \longrightarrow \mathbb{N}$ time constructible. These two definitions are equivalent with a possible delay in time f when implemented in a Turing machine.

However, in general, we may not have complete access to such weight so *queries* should be made. And here is why a real number is in fact an oracle. *It may encode noncomputable information.*[6]

Theorem 1.4.2 *(a) A system* Σ *with weights in* $\mathbb{Q} \cup \{r\}$*,* $r \in [0,1]$*, working in time* t *can be simulated by a Turing machine with a Davis's oracle* O_r *in time* $O(p(t))$*,* p *polynomial; (b) A Turing machine* M *with a Davis's oracle* O_r *[*O_r^f*] working in time* t *can be simulated by a neural net* Σ *with weights in* $\mathbb{Q} \cup \{r\}$*,* $r \in [0,1]$ *working in time* $O(p(t))$ *[*$O(f + p(t))$*].*

Proof (a) Build the following Davis's oracle to later retrieve the bits of r required for the simulation of Σ.

$$O = \{\langle 0^n, \tilde{r}\rangle : n \in \mathbb{N}, \tilde{r} \text{ is a prefix of } r_{|Trunc_t(n))}\}$$

The following Turing machine can simulate Σ with atmost a delay of polynomial of t

```
procedure:
        begin
                input $z$;
                $n := Trunc_t(|z|)$;
                $\tilde{r} := \lambda$;
                for $i := 1$ to $n$ do
                    if $\langle 0^n, \tilde{r}0\rangle \in O$,
                    then $\tilde{r} := \tilde{r}0$;
                    else if $\langle 0^n, \tilde{r}1\rangle \in O$,
                    then $\tilde{r} := \tilde{r}1$;
                    else exit for
                end for;
                simulate $\Sigma$ replacing $r$ by $\tilde{r}$ with input $z$;
                output its result
        end
```

(b) The other way around can be proved by separating oracle calls of M from other computations. Since M works in time t, for an input of size n, $t(n) + 1$ is an upper bound of the size of the query words. Fix an input z of size n.

Let $\tilde{r}$ be the first $t(n) + 1$ bits of r and $\tilde{M}$ the Turing machine that receives $\langle z, \tilde{r}\rangle$ as input and mimics all the computations of M, substituting oracle calls by the following procedure:

[5] r can be regarded as a real number in $[0, 1]$.

[6] Depending on the computability degree of the real number, one can have a hierarchy of complexity of ARNN with real weights (see [4]).

procedure:

```
begin
    input query word $w$;
    if $w$ is a prefix of $\tilde{r}$,
    then switch to state $q_{yes}$;
    else switch to state $q_{no}$
end
```

Build Σ_2 that simulates $\tilde{M}$. Construct Σ_1 with two input lines, feeding in the input $(z0^\omega)$ and the validation line $(1^{|z|}0^\omega)$. Σ_1 consists on a binary BAM, a counter that counts $t(n)+1$ steps (by real time simulation of the Turing machine that witnesses the fact that $t+1$ is time constructible) to extract $\tilde{r}$ and a unit that saves the value of z and then export it together with $\tilde{r}$ when prefix retrieval is done. Connecting Σ_1 to Σ_2, we have the neural net Σ with all the weights rational except r, working in time $O(t+1+t)$, that is, $O(p(t))$, with p polynomial.

The case of O_r^f can be proved similarly, by changing the number of bits to be retrieved by the BAM to $f(n)+1$. This net will work in time $O(f+1+t)$, hence $O(f+p(t))$ for some p polynomial. ∎

Remark The proof of Theorem 1.4.2 part (b) is essential for the last two chapters of this work. We can resume it into two ingredients. The nature of the oracle permits us to make queries through a BAM subsystem. On the other hand, separating an oracle Turing machine into two parts

1. one that performs all the oracle calls and
2. $\tilde{M}$ that receives all the accepted query words (in this case, one prefix of a real number) and the input to mimic the original machine,

will allow us to construct a neural net that simulates it with two subsystems

1. Σ_1 that extracts a prefix of a real number (a BAM) and
2. Σ_2 that simulates M_2 in real time.

Corollary 1.4.3 *For all* $r \in [0,1]$, $ARNN^P[\mathbb{Q} \cup \{r\}] = P(O_r) = \bigcup_{p \text{ polynomial}} P(O_r^p)$.

We will be writing $ARNN^P[\mathbb{Q}](r)$ instead of $ARNN^P[\mathbb{Q} \cup \{r\}]$ when we want to emphasize that r is to be seen as an oracle.

By induction, we can conclude

Corollary 1.4.4 $ARNN^P[\mathbb{Q}](r_1)(r_2)\dots(r_n) = P(O_{r_1})(O_{r_2})\dots(O_{r_n})$.

Theorem 1.4.5 *These classes of sets coincides:*

1. $P/poly$
2. $\bigcup_{S sparse} P(S)$
3. $\bigcup_{T tally} P(T)$

4. $\bigcup_{r\in[0,1]} ARNN^P[\mathbb{Q}](r)$

5. $ARNN^P[\mathbb{R}]$

6. $\bigcup_{O \text{Davis's Oracle}} P(O)$

This is also true for the analogous non-deterministic classes.

Proof (1) = (2) = (3) is known by Structural Complexity Theory. Theorem 1.2.5 states that (5) = (1) and from the last corollary we have (4) = (6). (1) $\subseteq$ (4) is guaranteed by Theorem 1.2.2 and trivially (4) $\subseteq$ (5). ▮

1.5 Time protocols of Oracles

The working time of a standard oracle is constant by definition. This means that when it is called, no matter how long a query word is, the time taken to answer the query is the same, which is unrealistic when a physical experiment is incorporated as an oracle. Certainly, we can "solve" this by substituting every call of a given oracle O by the following subsystem:

```
procedure:
          begin
                input $w$;
                count $T(|w|) - 1$ steps;
                if $w \in O$, then 'yes'
                else 'no'
          end
```

where w is the query word and T a time constructible function. Instead, we include naturally an internal clock in the oracle that works in same time units as the Turing machine which it is attached to.

Definition 1.5.1 *An oracle O* ***works in time*** *T when in each call the oracle takes $T(|z|)$ steps in time to answer the query $z \in O$.*[7]

When T is a polynomial or an exponential, we say that the oracle works under *polynomial* or *exponential time protocol or access time*.

One interesting property of these oracles is that the number of calls is limited when a time bound is set. We could have used this to define these oracles. But we still prefer this definition as to approximate the nature of physical experiments as oracles and that of Davis's oracles implemented in ARNN.

We will be interested in Davis's oracles of polynomial and exponential access time. When no protocol is refered, the oracle is just a classical one, that is, one that answers to queries in one time step.

[7]T may not be time constructible.

1.6 Generalized nets – Davis's oracle revisited

Recall that the Davis's oracles aim to retrieve the digits of real numbers.[8] The retrieval process works in linear time by the use of the saturated sigmoid. When other activation functions are adopted, the time to retrieve the real bits of that weight might be longer when simulated.

Definition 1.6.1 *A neural net is said to be a **generalized net** (of dimension N with M input lines) if the activation function is given by $f = \sigma \circ \pi$ where $\pi : \mathbb{R}^{N+M} \longrightarrow \mathbb{R}^N$ is an affine map*[9] *and $\sigma : \mathbb{R}^N \longrightarrow \mathbb{R}^N$, called the **activation function**, has a bounded range and is locally Lipschitz, that is, for each $\rho > 0$, there is a constant C such that for all x_1 and x_2 in the range of σ,*

$$|x_2 - x_1| < \rho \ \Rightarrow \ |\sigma(x_2) - \sigma(x_1)| \leq C|x_2 - x_1|$$

*where $|\cdot|$ is the Euclidean norm. The output units are chosen within the N processors and two **decision thresholds** $\alpha < \beta$ are set to be interpreted as 0 if an output is less than or equal to α and 1 if an output is greater than or equal to β.*

Definition 1.6.2 *A function f is **s-approximable in time** T if there is a Turing machine that computes $f_{|s(n)}$ in time $T(n)$ given input of total size n. When the function s is not explicitly mentioned, we are considering $s(n) = n$.*

Suppose now a neural net with the architecture of Theorem 1.4.2, substituting the activation function of the unit with the real weight r by f, a function approximable in time T, giving rise to a generalized processor net D. By incorporating a subsystem that simulates the Turing machine that computes f instead of Σ_1, given the first n bits of r we can retrieve the first n bits of $f(r)$ in $T(n)$ steps. [16] shows that we can then simulate D by a Turing machine (and so, by a neural net).

Definition 1.6.3 *A **generalized Davis's oracle** is a Davis's oracle $O_{f,r}$, a subset of $Prefix(f(r))$, where $r \in [0,1]$ and f is a function approximable in time T. The time protocol of this oracle is T.*

Example The trivial case is the saturated sigmoid

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

For $x \in [0,1]$, we have that $\sigma(x) = x$. In fact, $O_{\sigma,r} = O_r$ is just a Davis's oracle with linear access time.

[8]The encoding used will be omitted when it is not relevant to be mentioned.

[9]Although in the article [16] π is a generic polynomial in $N + M$ variables with real coefficients, we are interested only in affine maps.

Example Another example is the family of anayltic sigmoid

$$\sigma_a^k(x) = \frac{2}{1+e^{-kx}} - 1.$$

Its computation takes exponential time. That is, $O_{\sigma_a^k, r}$ is a Davis's oracle with exponential access time.

Example Suppose that the value $f(r)$ is given for a function f with right inverse, say f^{-1}, approximable in time T. Then, we can first retrieve n bits of $f(r)$ and then feed it into the Turing machine that computes n bits of $r = f^{-1}(f(r))$ in time T. This is a Davis's oracle with access time $O(n+T)$.

They can be seen as a blackbox with the following two components:

- A Davis's oracle O_r;
- A Turing machine M_f that approximates f in time T.

This oracle can be easily implemented in an ARNN, done as follows:

- A BAM subsystem to extract the bits of r;
- A subsystem simulating M_f in real time.

It takes $O(n+T(n))$ to compute the first n bits of r. Languages decided by these nets will be denoted by $ARNN[\mathbb{Q}](f, r)$.

Clearly they do not fit in the definition of a standard oracle. As stated before, an internal clock is *naturally* implemented into them, so we can control the time they take to answer a given query. By choosing a function f approximable in polynomial time and exponential time, we obtain respectively a *Davis's oracle with polynomial* and *exponential access time* and also their simulations in ARNN.

From now on, our goal is to present some relativization results. Two families of physical experiments will be emphasized and we will see that they are strongly related to Davis's oracles with polynomial and exponential access time.

Chapter 2

Relativization in ARNN: Oracles with polynomial access time

The main goal of this chapter is to show the positive relativization result for rational ARNN with one real weight, regarding this weight as a Davis's oracle with polynomial access time. Along this path, we will introduce the Scatter Machine Experiments as a class of physical oracles that simulate the extraction of bits from a real number.

2.1 Scatter Machine Experiments

In the description of a probabilistic Turing machine, it is common to integrate the coin flipping experiment done in one step as a way to choose between two branches with probability 1/2. A natural extension of this might be implementation of physical experiments as oracles in a Turing machine. This has been extensively explored by E. Beggs, J. Costa and J. Tucker in [5]. One of the experiments that has been analysed is the *Scatter Machine Experiment*.

The scatter machine experiment (SME) consists on firing a particle with a fixed velocity from a cannon in a position x given as input, projecting it over a surface with a wedge of 45° in a fixed position r. On each side of the wedge set one box, so that one of the two boxes collects the particle after reflecting on the surface (see Fig. 2.1). The goal is to approximate the position of the wedge with the position of the cannon given by the Turing machine using *bisection method*. We briefly explain this in the next paragraph.

Let r be a real number in $[0, 1]$. At step 1, compare r with $\frac{1}{2}$. In step 2, if $r < \frac{1}{2}$, then compare r to $\frac{1}{4}$. If $r > \frac{1}{2}$, compare r to $\frac{3}{4}$. And so on. In fact, the extraction can be done by subtracting r to a dyadic rational, that is, a rational of the form $\frac{m}{2^n}$. If the difference δ is less than 0, then $\sigma(\delta) = 0$. If $\delta > 0$, by iterating $\sigma(2\,\cdot)$ to δ, in some time we will obtain the value 1. In the case where r is a dyadic rational, this extraction will take forever since δ will be 0 in some step. We will exclude dyadic rationals in our analysis.[1]

As an oracle, denoted by $SME(r)$, it works as follows:

1. When the Turing machine is in the *query state*, the query tape contains x, the position of the cannon.

[1]This can also be solved by admiting three answer states after setting a time protocol, q_{yes}, q_{no} and $q_{\text{out of time}}$.

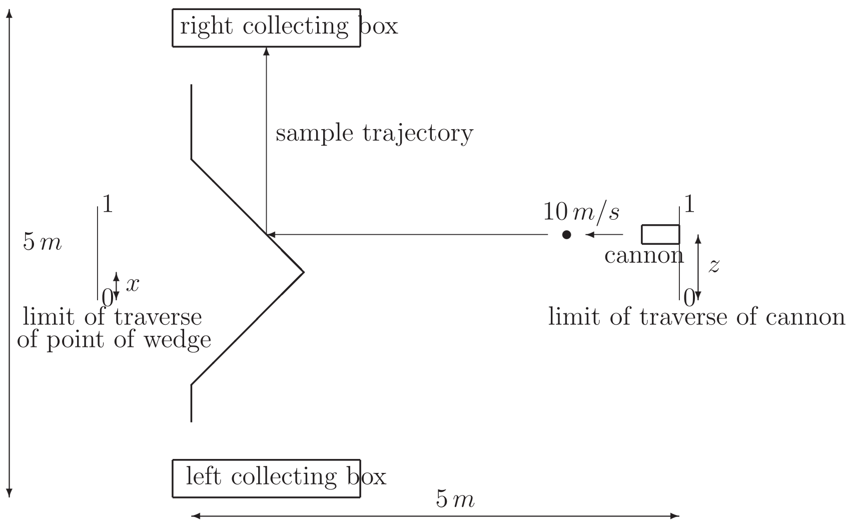


Figure 2.1: Example of a scatter machine experiment.

2. The cannon is set to the given position x (we assume that this can be done without error)[2] and fires the particle. If the particle is collected by the left box, i.e., $x > r$, then proceed to state *no*, otherwise proceed to *yes*.

Connecting a Turing machine to such experiments running in real time as oracles implies automatically the existence of an internal clock that is somehow related to the internal clock of the Turing machine. For this experiment, we assume the protocol of polynomial time (in this case we write $SME_p(r)$). This will give an upperbound of the number of queries, denoted by Q:

Definition 2.1.1 *Let M be a non-deterministic Turing machine equipped with an oracle O. We denote by $Q(M, z, O)$ the set of queries made by M in all branches of the computation tree when z is given as input. When there is no confusion, we write $Q(M, z, r)$ when the oracle is $SME_p(r)$.*

Definition 2.1.2 *Let Σ be a non-deterministic rational neural net with a real weight r. We denote by $Q(\Sigma, z, r)$ the set of queries made by Σ to r in all the branches of the computation tree when z is given as input.*

Proposition 2.1.3 *Let A be a set decided by a non-deterministic Turing machine M clocked in polynomial time p equipped with $SME_p(r)$. Then there is a non-deterministic Turing machine $\tilde{M}$ working in polynomial time querying the same oracle atmost a polynomial number of times. In fact, $|Q(\tilde{M}, z, r)| \leq p(|z|) + 1$.*

[2]We call it *error-free*. The SME is *error-prone* with fixed precision if there is a value $\varepsilon > 0$ such that the cannon can be set only to within a given precision ε.

Proof Given input z, a word written in the query tape is of length bounded by $p(|z|)$, since M works in polynomial time. So we only need the first digits of r. Let $\tilde{r}$ be the first $p(|z|)+1$ bits of r. We first build a Turing machine M_1 extracting the first $p(|z|)+1$ digits through the scatter machine using bisection method. Then, attach this machine to M_2, with the same description of M, substituting the oracle call by:

```
procedure:
        begin
            input $w$;
            for $i := 1$ to $|w|$ do
                if $w_i > \tilde{r}_i$, then 'no'
                else if $w_i < \tilde{r}_i$, then 'yes';
            proceed to 'no'
        end
```

This new machine $\tilde{M}$ still works in polynomial time. As the queries were all done at start, the total number of queries is $|Q(\tilde{M}, z, r)| \leq p(|z|)+1$. ▮

The physical oracle $SME_p(r)$ is a Davis's oracle of polynomial access time by definition. We will see how this will lead to the following relativization result:

Theorem 2.1.4 *$P = NP$ if and only if, for all $r \in [0,1]$,*

$$P(SME_p(r)) = NP(SME_p(r)).$$

2.2 Proof of Relativization result for Davis's oracles of polynomial time access

Proposition 2.2.1 *Let A be a set decided by a non-deterministic Turing machine M clocked in polynomial time p, equipped with a Davis's oracle O of polynomial time access. Then, there is a non-deterministic Turing machine $\tilde{M}$ working in polynomial time querying the same oracle a polynomial number of times during a computation. In fact, $|Q(\tilde{M}, z, O)| \leq p(|z|)+1$.*

Proof Suppose without loss of generality that $|\Gamma| = 2$. Let $O = Prefix(r)$ for a $r \in \{0,1\}^\omega$ where the extraction of the mth bit takes polynomial time $q(m)$.

Since M is working in polynomial time p, given an input z, the size of the queries does not exceed $p(|z|)$. So, only a polynomial number of bits of r is needed. We apply the bisection method.

Consider M_1 the machines that extracts the first $p(|z|)+1$ of r given input z:

```
procedure:
        begin
            input $z$;
            $\tilde{r} := \lambda$
            for $i := 1$ to $p(|z|)+1$ do
                if $\tilde{r}0 \in O$, then 'no'
                else if $\tilde{r}1 \in O$, then 'yes'
        end
```

This subsystem takes polynomial time $q(p(|z|))$ to perform the computations.[3] Build M_2 with the same description as M, substituting each oracle call by the following procedure:

```
procedure:
    begin
        input $w$;
        for $i := 1$ to $|w|$ do
            if $w_i > \tilde{r}_i$, then 'no'
            else if $w_i < \tilde{r}_i$, then 'yes';
        proceed to 'no'
    end
```

where w is the query word. M_2 works in polynomial time and so is $\tilde{M}$ which consists in attaching machine M_1 to M_2. The total number of oracle calls is indeed polynomial on the size of the input. ∎

Remark Note that it is trivial that the classes $P(O_r)$ and $P(O_r)_p = \bigcup_{f \text{approx. in p. time}} P(O_{f,r})$ coincides (implying

$$P/poly = \bigcup_{O_r \text{with p.a.t.}} P(O_r)$$

where p.a.t. abbreviates polynomial access time and O_r are Davis's oracles).

By simulating M_1 with a BAM and M_2 by the real time simulation of Turing machines by neural nets, we have the following corollary:

Corollary 2.2.2 *Let A be a set decided by a non-deterministic rational neural net Σ with a real weight r clocked in polynomial time p. Then, there is a non-deterministic neural net $\tilde{\Sigma}$ working in polynomial time querying r a polynomial number of times during a computation. In fact, $|Q(\tilde{\Sigma}, z, r)| \leq p(|z|) + 1$.*

Note that M has been composed into a machine that exclusively do the oracle calls, M_1, and the other without querying the oracle, M_2. This was used in the proof of Theorem 1.4.2.

Theorem 2.2.3 *(Positive relativization) $P = NP$ if and only if, for any Davis's oracle O with polynomial access time,*

$$P(O) = NP(O).$$

Proof Suppose that for all polynomial Davis oracles O, we have $P(O) = NP(O)$. By choosing $r = \frac{1}{3}$, O_r is computable in constant time. Hence

$$P = P(O_r) = NP(O_r) = NP$$

Now, assume $P = NP$. Let A be a set decided by a non-deterministic Turing machine M working in polynomial time p with a Davis's oracle O with polynomial time access implemented.

[3]This is indeed the simulation by a Turing machine of the bit extraction process done by a scatter machine and the BAM subsystem in an ARNN.

Construct M_1 and M_2 as in Proposition 2.2.1. Note that M_1 is a deterministic Turing machine with oracle O and M_2 is non-deterministic without oracle. Since $P = NP$, there is a deterministic Turing machine $\tilde{M}_2$ that decides the same set as M_2 in polynomial time. Connecting M_1 to $\tilde{M}_2$, a new Turing machine is built. It is deterministic, works in polynomial time and has implemented the same oracle O. ▮

By repeating the proof for particular restrictions we have:

Theorem 2.2.4 *$P = NP$ if and only if, for all $r \in [0, 1]$,*

$$P(SME_p(r)) = NP(SME_p(r)).$$

Theorem 2.2.5 *$P = NP$ if and only if, for any $r \in [0, 1]$ and f approximable in polynomial time,*

$$P(O_{f,r}) = NP(O_{f,r}).$$

As particular cases:

Theorem 2.2.6 *$P = NP$ if and only if, for any $r \in [0, 1]$ and f linearly approximable in polynomial time,*

$$ARNN^P[\mathbb{Q}](f, r) = ARNN^{NP}[\mathbb{Q}](f, r).$$

Theorem 2.2.7 *$P = NP$ if and only if, for any $r \in [0, 1]$,*

$$ARNN^P[\mathbb{Q}](r) = ARNN^{NP}[\mathbb{Q}](r).$$

Note that the left hand side of the equivalence of all these results claims that, not only the whole class will collapse, each deterministic subclass will coincide with its non-deterministic correspondent. The difference from the trivial result

$$P = NP \text{ if and only if } P/poly = NP/poly$$

should be emphasized.

Theorem 2.2.8 *The following propositions are equivalent:*

1. $P = NP$
2. $ARNN^P[\mathbb{Q}] = ARNN^{NP}[\mathbb{Q}]$
3. $ARNN^P[\mathbb{R}] = ARNN^{NP}[\mathbb{R}]$

We can conclude that the class of Davis's oracles with polynomial time access is indeed a restricted class of oracles, since it is known that there are oracles A and B such that $P(A) = NP(A)$ and $P(B) \neq NP(B)$ (the Baker-Gill-Solovay Theorems). The negative relativization for Davis's oracles with polynomial time access is still an open problem. In contrast, this will be proved to be true for those with exponential time access in the next chapter.

Chapter 3

Relativization in ARNN: Oracles with exponential access time

For Davis's oracles with polynomial access time, we only have the positive relativization result. We will show that there is a positive relativization of the exponential access time oracle but also a negative one. First we will refer to another physical oracle: the Collider Machine Experiments.

3.1 Collider Machine Experiments

Another physical experiment explored is the *Collider Machine Experiments* (CME). It consists on approximating the mass of a given particle by colliding it with another one with a dyadic mass using the bisection method. This is a one dimensional experiment (see Fig. 3.1). Assume that we can move a particle in any given speed and that there is preservation of momentum and kinetic energy.

Let $\mu = r$ be the mass to be approximated, located at the origin, and m the mass of the test particle. Suppose that the test particle is launched with speed v_0 from position $-P$, for some $P > 0$. If $m < r$, the proof particle will return to $-P$ in some time. If $m > r$, then it will move towards the direction of $+P$ after collision.[1]

In the following paragraphs, the time taken by these experiments will be derived.

The momentum and the kinetic energy of a particle with mass m is given by mv and $\frac{1}{2}mv^2$. By preservation of energy and momentum of the system, we have:

$$\begin{aligned} mv_0 &= mv_m + rv_r, \\ \frac{1}{2}mv_0^2 &= \frac{1}{2}mv_m^2 + \frac{1}{2}mv_r^2 \end{aligned}$$

Solving in respect to v_m and v_r:

$$v_m = \frac{m-r}{m+r}v_0$$

[1]Since we considered that r is not a dyadic rational, the extraction is done in finite time.

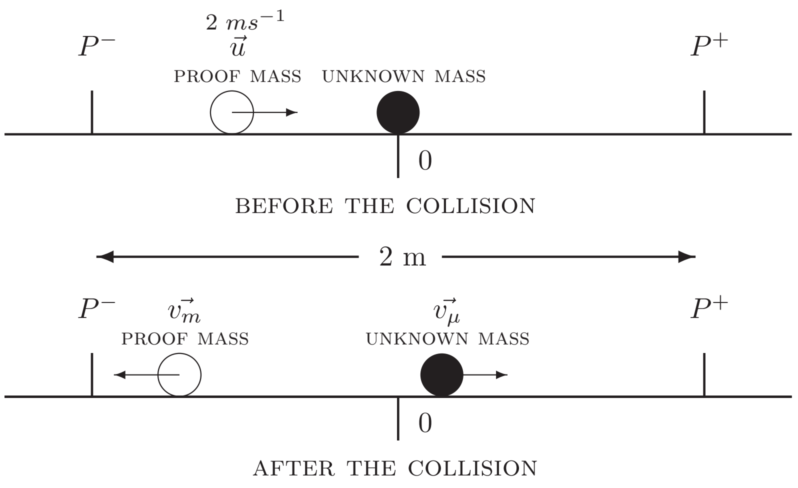


Figure 3.1: Example of a collider machine experiment.

By the last equation, we can conclude that the time taken by the experiment is given by

$$\begin{aligned} T &= \left(\frac{2r}{|m-r|} \pm 1\right)\frac{P}{v_0} \\ &\sim \frac{K}{|m-r|} \end{aligned}$$

If $|m - r| < 2^{-n}$, the time needed will be of exponential order. For this reason, when we use the collider machine experiment as physical oracle, the protocol is of either exponential or unbounded time. To denote such experiments, we write $CME_e(r)$ or $CME_\infty(r)$, respectively. We will consider the exponential protocol. The unbounded time protocol will only be relevant when we allow dyadic rational oracles. An extensive study of its properties is in [5].

In the same article, the following theorems are proven:

Theorem 3.1.1 *$P = NP$ if and only if, for all $r \in [0, 1]$,*

$$P(CME_e(r)) = NP(CME_e(r)).$$

Theorem 3.1.2 *$P \neq NP$ if and only if, for all $r \in [0, 1]$,*

$$P(CME_e(r)) \neq NP(CME_e(r)).$$

We will provide their proofs in the next section.

3.2 Proof of Relativization results for Davis's oracles of exponential time access

Both $CME_e(r)$ and the analytic oracles are members of a larger class of oracles: the Davis's oracles with exponential access time. As in the polynomial case, we have the following results which will not be proved since their proofs are essentially the same.

Proposition 3.2.1 *Let A be a set decided by a non-deterministic Turing machine M clocked in polynomial time with an exponential Davis's oracle implemented in it. Then, there is a non-deterministic Turing machine $\tilde{M}$ also working in polynomial time querying the same oracle only a logarithmic number of times, where the total number of query words in all branches is logarithmic.*

Denote by $P_\ell(O)$ the class of languages decided by Turing machines working in polynomial time equipped with an oracle such that, for inputs of size n, only a logarithmic number on n of queries are admitted to be made. The last proposition showed that

Corollary 3.2.2 *For all Davis's oracles O_r with exponential access time, $P(O_r) = P_\ell(O_r)$.*

In particular,

Corollary 3.2.3 *For all Davis's oracles O_r and f approximable in exponential time, $P(O_{f,r}) = P_\ell(O_{f,r})$.*

Definition 3.2.4 *We denote by $\mathcal{F}*$ the set of prefix functions with size limited by a function in $\mathcal{F}$. That is, if $f \in \mathcal{F}*$, then for all n, $f(n)$ is a prefix of $f(n+1)$ and $|f(n)| \leq g(n)$ for some function $g \in \mathcal{F}$.*

Lemma 3.2.5 *For all Davis's oracle O with exponential access time, $P(O) \subseteq P/log* \subseteq P/log$.*

Remark The first inclusion can be easily extended to

$$P/log* = \bigcup_{O_r \text{with e.a.t.}} P(O)$$

where e.a.t. abbreviates exponential access time and O_r are Davis's oracles.

For this class of oracles we have also the negative relativization result, which is still an open problem for the polynomial ones.

Theorem 3.2.6 *(Positive relativization) $P = NP$ if and only if, for all Davis's oracles O with exponential time access,*

$$P(O) = NP(O).$$

Theorem 3.2.7 *(Negative relativization) $P \neq NP$ if and only if, for all Davis's oracles O with exponential time access,*

$$P(O) \neq NP(O).$$

The proof of the first result is omitted since it applies the same techniques as in the proof of Theorem 2.2.3. We shall prove the second one. For this we will need the following lemma well known in Structural Complexity theory [3]:

Lemma 3.2.8 *If $SAT \in P/log$, then $P = NP$.*

Proof (of Theorem 3.2.7) If the statement on the right hand side is true, then for the particular case of $O_{\frac{1}{3}}$, we have that $P \neq NP$ (see the correspondent case for polynomial access time).

Suppose now that $P \neq NP$. Let O be a Davis's oracle with exponential time access. By Lemma 3.2.8, $SAT \notin P/log$. In particular, $SAT \notin P(O)$ by Lemma 3.2.5. As known in Structural Complexity theory, SAT is in NP, which implies that SAT is in $NP(O)$ (without any oracle call). This proves that $P(O) \neq NP(O)$. ▮

As particular cases, we have:

Theorem 3.2.9 *$P = NP$ if and only if, for all $r \in [0, 1]$,*

$$P(CME_e(r)) = NP(CME_e(r)).$$

Theorem 3.2.10 *$P \neq NP$ if and only if, for all $r \in [0, 1]$,*

$$P(CME_e(r)) \neq NP(CME_e(r)).$$

Theorem 3.2.11 *$P = NP$ if and only if, for all $r \in [0, 1]$ and f approximable in exponential time,*

$$P(O_{f,r}) = NP(O_{f,r}).$$

Theorem 3.2.12 *$P \neq NP$ if and only if, for all $r \in [0, 1]$ and f approximable in exponential time,*

$$P(O_{f,r}) \neq NP(O_{f,r}).$$

Transposing these results to neural nets,

Theorem 3.2.13 *$P = NP$ if and only if, for all $r \in [0, 1]$ and f approximable in exponential time,*

$$ARNN^{P}[\mathbb{Q}](f, r) = ARNN^{NP}[\mathbb{Q}](f, r).$$

Theorem 3.2.14 *$P \neq NP$ if and only if, for all $r \in [0, 1]$ and f approximable in exponential time,*

$$ARNN^{P}[\mathbb{Q}](f, r) \neq ARNN^{NP}[\mathbb{Q}](f, r).$$

Conclusion

We provided a formalization of the idea of a real weight internalized in a rational ARNN regarded as a Davis's oracle, concretizing the criticism of Martin Davis in a constructive point of view. The simulation of a rational neural net with one real weight by a Turing machine with a Davis's oracle defined by that real weight shows that, not only the whole class of $ARNN^P[\mathbb{R}]$ coincide with $P/poly$, but there is also an inductive correspondence between the subclasses by incorporating the real weights one by one. This creates a hierarchy in $P/poly$, where each class can be represented by a finite set of real weights used, that is, sets of the form

$$\{r_1, r_2, \ldots, r_n : r_i \in \mathbb{R}, \forall 1 \leq i \leq n\}.$$

Taking into account time protocols of oracles, we proved that SME and CME are particular cases of Davis's oracles with polynomial and exponential access time, respectively. Furthermore, the famous problem $P = NP$ positively relativizes for Davis's oracles with both access times. We immediately conclude that Davis's oracles constitute a restricted class of oracles. In respect to real neural nets working in polynomial time, we conclude that they are in fact a restricted computational model. In resume, we have the following classification table:

Set of weights	Time restriction	Protocol	Computational power
$\mathbb{Z}$	none	none	Regular languages
$\mathbb{Q}$	none	none	Recursive languages
$\mathbb{Q}$	t	none	DTIME(t)
$\mathbb{Q} \cup \{r_1\}$	polynomial	exponential	$P_\ell(O_{r_1})$
$\vdots$			$\vdots$
$\mathbb{Q} \cup \{r_1, \ldots, r_n\}$	polynomial	exponential	$P_\ell(O_{r_1}) \ldots (O_{r_n})$
$\vdots$			$\vdots$
$\mathbb{R}$	polynomial	exponential	$P/log*$
$\mathbb{Q} \cup \{r_1\}$	polynomial	polynomial	$P(O_{r_1})$
$\vdots$			$\vdots$
$\mathbb{Q} \cup \{r_1, \ldots, r_n\}$	polynomial	polynomial	$P(O_{r_1}) \ldots (O_{r_n})$
$\vdots$			$\vdots$
$\mathbb{R}$	polynomial	polynomial	$P/poly$
$\mathbb{R}$	none	none	All languages

Table 3.1: A more refined classification by computational power of ARNN.

For the case of exponential access time, we also have the relativization of $P \neq NP$. This result is still an open problem for the polynomial case. Once proven (or disproved) for SME, its proof might be adapted to the case of Davis's oracles of polynomial access time and, in particular, real ARNN working in polynomial time.

Returning to more "grounded" issues, there are still simple questions that seem not simple to be answered. For instance, a very rewarding work would be a good classification of functional properties of neural nets. Within the logic system described by McCulloch and Pitts in [11], it is possible to realize a given logical function. But giving a description of *how* one neural network performs its computation is not trivial. Most of the known work refers to layered neural nets (for example, BAM).

Computational approach did not show to be the best single way to understand our brain and to explain how it works. But it turned out to be a relatively good tool to functionally map our brain and, at the same time, provided a very interesting biologically inspired computational model that uniformly classifies complexity classes in a surprising way.

Bibliography

[1] James A. Anderson. *An Introduction to Neural Networks.* Massachusetts Institute of Technology, 1995.

[2] José Luis Balcázar, Josep Días, and Joaquim Gabarró. *Structural Complexity I.* Springer-Verlag, 2nd edition, 1988, 1995.

[3] José Luis Balcázar, Josep Días, and Joaquim Gabarró. *Structural Complexity II.* Springer-Verlag, 1990.

[4] Edwin Beggs, José Félix Costa, Bruno Loff, and John V. Tucker. Oracles and advice as measurements. In Cristian S. Calude, José Félix Costa, Rudolf Freund, Marion Oswald, and Grzegorz Rozenberg, editors, *Unconventional Computation (UC 2008)*, volume 5204 of *Lecture Notes in Computer Science*, pages 33–50. Springer-Verlag, 2008.

[5] Edwin Beggs, José Félix Costa, and John V. Tucker. Comparing complexity classes relative to physical oracles. 2009. Technical Report.

[6] Jack Copeland and Diane Proudfoot. Alan Turing's forgotten ideas in Computer Science. *Scientific American*, 280:99–103, 1999.

[7] Martin Davis. The myth of hypercomputation. In Christof Teuscher, editor, *Alan Turing: the life and legacy of a great thinker*, pages 195–212. Springer, 2006.

[8] Robin Gandy. The confluence of ideas in 1936. In Rolf Herken, editor, *The universal Turing machine: a half-century survey*, pages 55–111. Oxford University Press, 1988.

[9] S. Haykin. *Neural Networks: A Comprehensive Foundation.* MacMillan College Publishing, 1994.

[10] Hava T. Siegelmann João Pedro Neto and José Félix Costa. Symbolic Processing in Neural Networks. *Journal of the Brazilian Computer Society*, 8(3):58–70, 2003.

[11] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Journal of Mathematical Analysis and Applications*, 5:115–133, 1943.

[12] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Engelwood Cliffs, NJ, 1967.

[13] James Nyce. Nature's machine: mimesis, the analog computer and the rhetoric of technology. In Ray Paton, editor, *Computing with Biological Metaphors*. Chapman & Hall, 1994.

[14] Mark Schlatter and Ken Aizawa. Walter Pitts and "A Logical Calculus". *Synthese*, 162:235–250, 2008.

[15] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural networks. pages 331–360, 1992.

[16] Hava T. Siegelmann and Eduardo D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360, 1994.

[17] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural networks. *J. Comp. Syst. Sciences*, 50(1):132–150, 1995.

[18] Eduardo D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Springer-Verlag, 1990.

Appendix A

Basic definitions

Here we will recall the definition of some abstract devices relevant for Part I.

Definition A.0.1 *An* ***alphabet*** Σ *is a non-empty finite set. Its elements are called* ***letters*** *or* ***symbols****.*

Definition A.0.2 Σ^* *denotes the set of* ***words****, that is, finite sequences of letters in* Σ*. By convention, the empty word* λ *is in* Σ^**.* Σ^+ *denotes the set of non-empty finite words.*

Definition A.0.3 *A* ***set*** *or a* ***language*** *is a subset of* Σ^+*.*

Definition A.0.4 *An* ***encoding*** *of an alphabet* Σ *over another alphabet* Γ *is*

1. *an injective function* $\Sigma \longrightarrow \Gamma$ *if* $|\Sigma| \leq |\Gamma|$
2. *an injective function* $\Sigma \longrightarrow \Gamma^n$ *if* $|\Sigma| > |\Gamma|$*, for some* n *such that* $|\Sigma| \leq |\Gamma|^n$

Definition A.0.5 *A* ***finite automaton*** *is a tuple* $M = \langle Q, \Sigma, \delta, q_0, \mathcal{F} \rangle$ *such that:*

- Q *is a finite non-empty set (elements of* Q *are called* ***states****)*
- Σ *is an alphabet*
- $\delta : Q \times \Sigma \longrightarrow Q$ *called the* ***transition function***
- $q_0 \in Q$*, called the* ***initial state***
- $\mathcal{F} \subseteq Q$ *is the set of* ***accepting*** *or* ***final states***[1]

Definition A.0.6 *Let* $M = \langle Q, \Sigma, \delta, q_0, \mathcal{F} \rangle$ *be a finite automaton and let* $w = w_1 w_2 \dots w_n \in \Sigma^*$*. We say that* M ***accepts*** w *if there is a sequence of states* $r_0, r_1, \dots, r_n$ *such that*

1. $r_0 \equiv q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1},\ 0 \leq i < n$

[1]It is a convention that M will switch off itself once attained some element of $\mathcal{F}$.

3. $r_n \in \mathcal{F}$

If a word is not accepted, we say that it is ***rejected****.*

Definition A.0.7 *A (1-taped)* ***Turing machine*** *M is a tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\rangle$ where:*

- *Q is a finite non-empty set (of* ***control states****);*
- *Σ is an alphabet (of* ***input****);*
- *Γ is an alphabet (of* ***tape****) such that $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma \setminus \Sigma$;*
- *$q_0, q_{accept}, q_{reject} \in Q$ (the* ***initial state****,* ***accepting state*** *and* ***rejecting state****) all three distinct from each other;*
- *$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{R, L\}$ a partial function, not defined for q_{accept} and q_{reject} (the* ***transition function****).*

The machine stops when q_{accept} or q_{reject} is attained.

The sequence of R and L given after some iterations of δ gives us the position of what we call the *reading head*. When the machine is in state q and a word $w = w_1w_2 \in \Gamma^+$ is written in the tape with the reading head in the start of w_2, we denote this *configuration* by w_1qw_2.

Definition A.0.8 *A* ***non-deterministic Turing machine*** *is a machine similar to a deterministic one where the transition function is*

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{R, L\}).$$

Definition A.0.9 *A Turing machine with k* ***tapes*** *is defined analogously as a 1-taped Turing machine, where*[2]

$$\delta : Q \times \Gamma^k \longrightarrow \mathcal{P}(Q \times \Gamma^{k-1} \times \{R, L\}^k).$$

Definition A.0.10 *A Turing machine M* ***accepts*** *a word $w \in \Sigma^+$ if there exists a sequence $c_0 \dots c_k$ of configurations, $k \geq 1$, such thtat:*

1. *$c_0 = q_0w$;*
2. *c_{i+1} is obtained from c_i by the function δ, $0 \leq i < k$;*
3. *c_k contains q_{accept}.*

Similarly, a word is ***rejected*** *when c_k contains q_{reject}*

Note that a word that is not accepted is not necessarily rejected.

Definition A.0.11 *A total function $f : \mathbb{N} \longrightarrow \mathbb{N}$ is said to be* ***constructible in time*** *if there is a Turing machine M such that, for all $n \in \mathbb{N}$ and all inputs of size n, halts exactly in $f(n)$ steps.*

[2]When there are more than one tape, it is a convention to consider that one of the tapes is for the input and is not altered during computation.

Appendix B

An introduction to Cantor sets and Encoding Systems

Codifying finite information is a trivial task. But once we are dealing with information of infinite size, for instance, an infinite subset of $\{0,1\}^+$, we should be able to compact it in an efficient and "decodable" way.

Definition B.0.1 *Two sets are said to be* ***equipotent*** *if a bijection between them exists.*

Proposition B.0.2 *The set of infinite languages of the alphabet* $\{0,1\}$ *is equipotent to* $[0,1]$.

Proof If we organize the words in lexicographical order, it can then be enumerated by the natural numbers. Now, an infinite set of words can be fully characterized by coding, for instance, 1 in the n^{th} position if the n^{th} element belongs to it, 0 otherwise. ∎

Corollary B.0.3 *The set of infinite languages of the alphabet* $\{0,1\}$ *are equipotent to* $\mathbb{R}$.

When two sets are equipotent, there are encodings from one to another. When the encodings take values in $[0,1]$, there are two problems that naturally arise.

1. The problem of uniqueness of representation in a certain base n arises when we are dealing with expansions of the form

$$\sum_{i=1}^{+\infty} \frac{a_i}{n^i}$$

   where $a_i \in \{0,1,\ldots,n-1\}$ for all i and for some $k \in \mathbb{N}$, $a_i = n-1$ for all $i \geq k$. In this case, the other representation is

$$\frac{a_{k-1}+1}{n^{k-1}} + \sum_{i=1}^{k-2} \frac{a_i}{n^i}$$

   For example, in base 10, $0.09999\ldots = 0.1$.

2. The problem of determining if a number is 0.

For problem 1, we should look for a representation where the choice of consecutive digits is not allowed. To explain this, let us show an illustrative example. Suppose we have the number 0.57999.... If we admit the use of the digit 8 in our encoding system, then another representation of this would be 0.58.

A set that arises by this choice is a *Cantor* subset of the interval. When the base of the encoding is n, we call it an n-Cantor subset, denoted $\mathcal{C}_n$, and the encoding an n-Cantor encoding.

Suppose that the not consecutive digits allowed are $\{a_1, \ldots, a_k\} \subset \{0, 1, \ldots, n-1\}$. The encoding takes values of the form

$$\bigcap_{i \geq 1} \left[\frac{a_i}{n^i}, \frac{a_i + 1}{n^i}\right)$$

It is easy to see that this construction will lead to a totally disconnect, perfect[1] and compact metric space with the inherited metric from the reals, that is, a *Cantor set* in the interval. Hence our denomination is sound in the topological sense. Proposition B.0.2 shows that Cantor sets are uncountable.

Example Families of Boolean circuits are encoded into real numbers in base 9 with digits within 0, 2, 4, 6 and 8. See Part II Section 1.3.

For problem 2, the empty word λ can be mapped to a special number, for example 0, and avoid the use of 0 in our encoding. This will imply that in base n

$$\sigma(nx)$$

tests if a word is non-empty (0 if x encodes λ and 1 otherwise).

Example The universal architecture for simulation of Turing machines accepts as inputs finite words in binary encoded into rational numbers in base 4, where the digits that are allowed are 1 and 3 (Part I Section 2.4.). The funtion *nonempty* evaluates if the word encoded is non-empty. Another encoding was used to accelerate the simulation, using base $10p^2$

$$[\alpha] = \sum_{i=1}^{|\alpha|} \frac{10p^2 - 1 + 4p(\alpha_i - 1)}{(10p^2)^i}.$$

Conversely, we require three properties from such a "good" encoding set in $[0, 1]$: it should be totally disconnected with respect to the induced topology of $\mathbb{R}$ so no interval belongs to it (this corresponds to the limited choice of digits in the n-Cantor system), uncountable to be equipotent to the set of infinite languages, recursively constructed in a "self-similar" fashion so we can have a base for representation. Any n-Cantor set is indeed homeomorphic to the usual Cantor set by equipping these sets with the following metric:

$$d(\alpha, \beta) = \sum_{i=1}^{+\infty} \frac{\delta(\alpha_i, \beta_i)}{2^i},$$

where $\delta(a, b)$ is 0 if $a = b$ and 1 otherwise, that is, the Kronecker's delta.

[1] A set is said to be *perfect* if every point is an accumulation point of the set, or equivalently, there is no isolated point.

Definition B.0.4 *A **good enconding** of a set $L \subseteq \Sigma^+$ is an injective map $L \longrightarrow \mathcal{C}$, induced by an encoding from Σ to the allowed digits in the expansion of elements in $\mathcal{C}$, where $\mathcal{C}$ denotes a Cantor subset in* $[0, 1]$.

Index