Università di Padova

Department of Physics "Galileo Galilei"

Master Thesis in Physics of Data

Convergent evolution of neuron architectures

Supervisors

Prof. Ricard Solé

Universitat Pompeu Fabra

Prof. Samir Simon Suweis

Università di Padova

Co-supervisors

Jordi Piñero

Michigan State University

Master Candidate

Pietro Malagoli

Student ID

2125711

Academic Year

2025-2026

Ai miei genitori

Abstract

A universal structural pattern of cognitive architectures is the presence of a class of fundamental units that display polarity (information is sent in one direction) and threshold response dynamics. Despite the potential diversity of design principles, this architecture suggests a convergent solution to the problem of sensing and processing information in natural systems. The purpose of this work is to explore this possibility by evolving artificial systems of connected units (cells) that start from a homogeneous network of linear elements, with the capability of developing a non-linear activation function.

Abstract

Un modello strutturale universale delle architetture cognitive è la presenza di una classe di unità fondamentali che mostrano polarità (le informazioni vengono inviate in una sola direzione) e dinamiche di risposta basate su una soglia. Nonostante la potenziale diversità dei principi di progettazione, questa architettura suggerisce una soluzione convergente al problema della rilevazione e dell'elaborazione delle informazioni nei sistemi naturali. L'obiettivo di questo studio è di esplorare questa possibilità, evolvendo sistemi artificiali di unità connesse (cellule) che partono da una rete omogenea di elementi lineari, con la capacità di sviluppare una funzione di attivazione non lineare.

Contents

List of Figures

List of Tables

List of acronyms

EA . . . . . Evolutionary Algorithm

GA . . . . . Genetic Algorithm

1

Introduction

In the last sentence of his historic book *On the Origin of Species*, Charles Darwin celebrates the imaginative power of evolution (Darwin 1859, 490):

> *"...from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved."*

Standing on the shoulders of such a giant, biologists today are challenging the idea that evolution can indeed give birth to *endless forms*. In fact, evolution has historically proposed analogous solutions to similar problems over and over, leading to the concept of convergent evolution[1].

Convergent evolution

Convergent evolution is the *independent* development of similar phenotypic characteristics in species of different lineages and it is an omnipresent theme in evolutionary biology. The recurrent evolution of flight is a common example as birds, bats, insects and pterosaurs have all independently developed the powerful capacity of flight [2]. However, convergent evolution has touched virtually every class of phenotypic development: from eye structure in cephalopods and vertebrates to C_4 photosynthesis in plants [3, 4].

Origins of cognition

When dealing with the issue of a dangerously unpredictable external environment, storing and accessing information about such outside world is crucial to reduce its uncertainty[5]. In this endeavor, evolution has exhibited a motif: neural systems have developed independently numerous times in history. Instances of such convergent evolution

can be found in, for example, Ctenophora (comb jellies), cephalopods and vertebrates, which, despite having evolved along separate branches, all exhibit comparable neural systems that employ neurons as their basic component[6, 7].

Is the neuron the only possible architecture?

Neurons are fundamental units of cognition that display polarity (information is sent in one direction) and threshold response dynamics. They are the building blocks of the nervous system, offering a way to process both external and internal inputs. Networks of neurons provide overwhelmingly advantageous emergent capabilities when dealing with a complex, ever-changing environment, such as learning new tasks, storing information (memories), and, to a certain extent, predicting the dynamics of the surroundings. Therefore, as expected, neurons are a frequently proposed medium for cognition and a prime example of convergent evolution[5, 6].
This thesis aims to construct a model for answering a natural question stemming from these previous observations: is the neuron, as a unit displaying polarity and threshold response dynamics, the only evolutionarily feasible candidate as the fundamental component for cognition?

2

Theoretical Background

In this chapter, some foundational concepts for the study are discussed synthetically. In the first section (Section 2.1), the notion of convergent evolution in biology is presented through its major historical contributions and some examples from the biosphere. In the second section (Section 2.2), the description focuses on the biological evolution of neurons as an instance of convergent evolution, which is the foundational idea from which this study originates. Finally, in the third section (Section 2.3), genetic algorithms are presented as a model of natural evolution, and a simple evolutionary algorithm is described, as an example of evolutionary computation.

2.1 Convergent evolution

How different would an alien biosphere be from the one developed on Earth? It is a widespread idea that extraterrestrial life would be close to inconceivable in comparison with life on Earth, and that the vastness of the universe grants a combinatorial size to the endless forms life could take.
Pondering the possibilities of the infinite causes the gaze to revert back to the past, in particular, to the origin. A way to make sense of the present is to re-run the tape of history and analyze the outcomes.
Stephen Jay Gould argued that, due to the historic nature of evolution, acting in such a way would produce completely different results, even when the same conditions are encountered again. His thesis was that contingency plays a major role in determining the path taken by evolution. He largely based its conclusions on the event of the Cambrian explosion, pointing out that a great part of the anatomical body plans (phyla) present in the Cambrian biosphere did not reach the present. He highlights that this occurred regardless of the lack of evidence to prove their selective disadvantages when compared to the phyla that later evolved into the present animal biosphere (e.g. *Pikaia*)[8].

Simon Conway Morris disputed Gould's thesis, pointing out that life is constantly kept on a close watch by numerous constraints, from natural selection to physical and chemical factors, narrowing its possibilities of expression. He argued that convergence is a dominant force in evolution and, given the same conditions, life will inevitably evolve towards an optimum configuration. In other words, not everything is possible to evolution.
An interesting geometric take on the concept of fundamental constraints in the evolution of biological systems was presented by David Raup: the morphospace, i.e. a multi-dimensional space enclosing all the combinations of the parameters regulating a certain phenotype of a given class of organisms. A key aspect emerging from studying the distribution of organisms in the morphospace is that some regions are densely populated while others are void. In Raup's example, only a fraction of the morphospace of possible shell coiling is occupied or, in other words, only a handful of possible types of shell coiling have been observed in nature, while the major fraction remains unobserved[9]. This anisotropic distribution of life in the morphospace is closely affiliated with the role of constraints and substantiates the thesis of convergent evolution[5].
Waddington argues that the constancy of the wild type, which is the form of an organism occurring in Nature under the forces of natural selection, is proof that developmental reactions are, in general, canalized (converged). In other words, the evidence that the wild type of an organism is much less variable in phenotype than the majority of its mutant forms is a validation of canalization. That is to say that the genotype can absorb a considerable amount of mutations before displaying any alteration in development. Furthermore, he advocates for robustness as a pragmatic argument for canalization: the convergent nature of evolution ensures the production of the standard, which is the optimal type in dealing with the hazards of the environment[10].
Finally, Stuart Alan Kauffman et al. state that the biosphere is the most complex system known in the universe and that evolution is intrinsically unpredictable due to the 'non-ergodic' nature of biology. In other words, the space of possible complex molecules is extremely vast, and the space of possible functions of complex molecules in biology is even greater, much larger than what would be feasible to explore for evolution in the lifetime of the universe. In their own words: "*It is true that most complex things will never 'get to exist'.*"[11].

Other than by the arguments presented in the previous paragraph, the thesis of convergent evolution is substantiated by numerous examples in the history of the biosphere. These instances of canalization in the developmental reactions span all of Nature, from

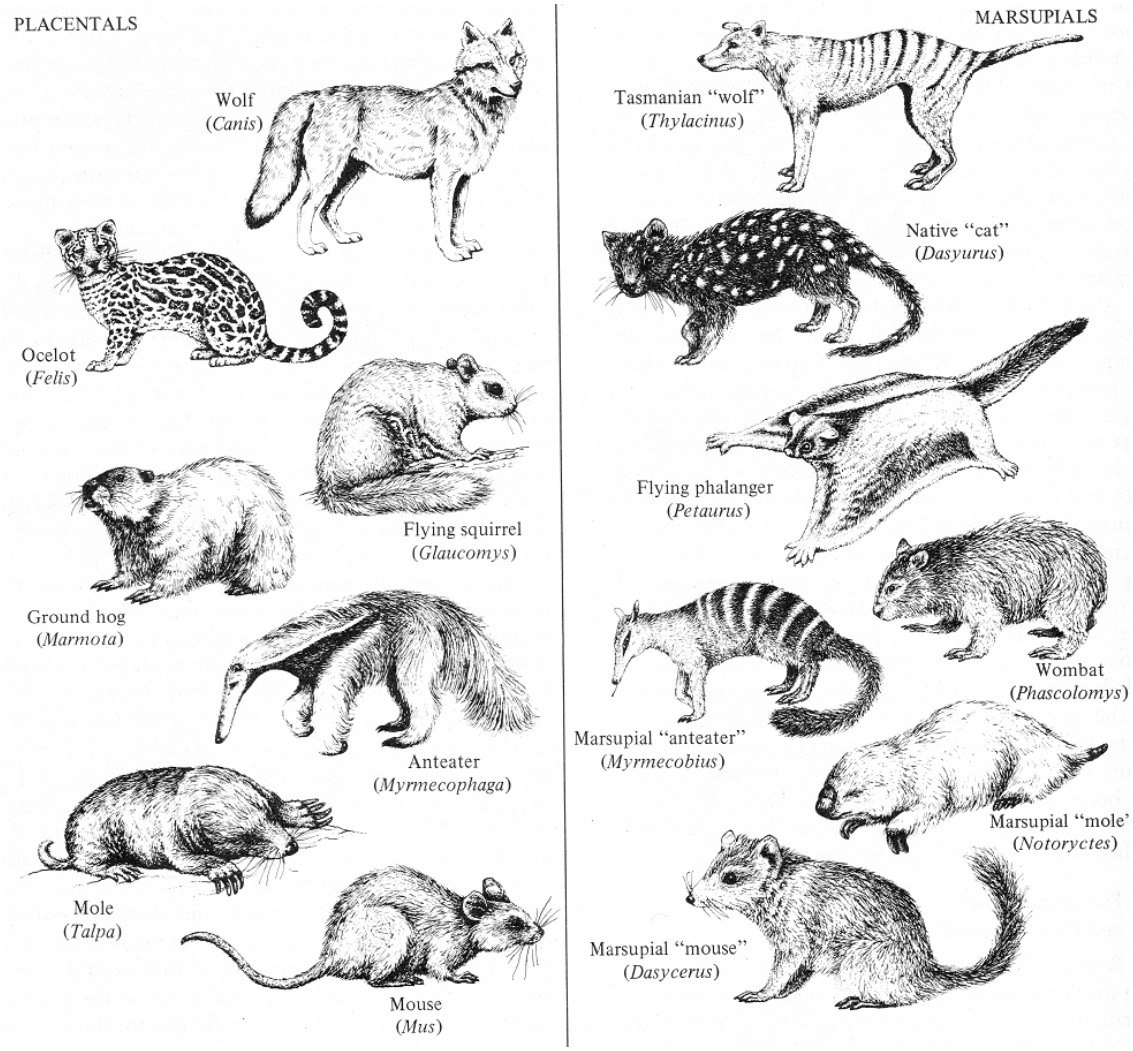

Figure 2.1: Example of convergent evolution in mammals and marsupials. Figure adapted from Begon *et al.*[12]

.

proteins to animals and plants, and at every level of complexity, from body plans to the development of intelligence.

One of the most commonly reported instances of convergent evolution is in the recurrent evolution of flight: birds and bats both have wings, but their structures exhibit major differences. Bats are the only mammals capable of sustained flight, and their wings are formed by a membrane that stretches between the arms and the fingers, while bird wings consist of feathers extending over the entire limb[2]. These dissimilarities suggest that wings in birds and bats have evolved independently, as an example of functional convergence. They are similar because they have undergone the same forces of natural

selection and developed the same answer to the common problem of locomotion[1, 13].

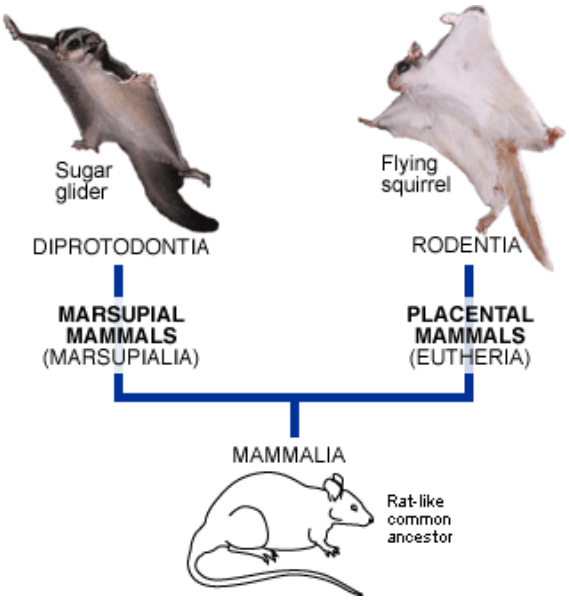

Figure 2.2: Phylogenetic tree of flying squirrels and sugar gliders (figure from "Analogy: Squirrels and Sugar Gliders". University of California Berkeley [14]).

Flying squirrels and sugar gliders have similarities in their body plans, above all gliding wings formed by skin stretching between their arms and legs. Both species utilize this trait to leap from heights. However, flying squirrels are placental mammals, while sugar gliders are marsupial mammals; these two species are only distantly related. The gliding mechanism is an analogous structure developed through a canalized process of natural selection (Figure 2.2)[14]. Figure 2.1 shows examples of homologies between various mammal and marsupial species.

Examples of convergent evolution can also be found outside of the animal world: C_4 photosynthesis has emerged independently in at least 60 plant lineages, despite its high complexity. Species characterized by C_4 carbon fixation represent the most productive plants on the planet, proposing a pragmatic argument for the convergent evolution of this trait[4].

Finally, this principle permeates down to the fundamental units of life: in fact, instances of convergent evolution are also found in protein structures [15].

2.2 Convergent evolution of neurons

Did neurons evolve multiple times?

Two scenarios of neuronal evolution are possible: polygenesis or monophyly. The former consists of the thesis that neurons evolved independently more than once during history, while the latter argues for a single origin of neurons.

The polygenesis hypothesis advocates for neurons to be an example of convergent evolution (or homology, in biogenetic lingo), with its major clue being the presence of

neuronal structures in ctenophores, and in cnidarians and bilaterians[6]. Ctenophores are a phylum of marine invertebrates commonly known as comb jellies, while cnidarians constitute the major part of the remaining marine invertebrates, with over 11,000 species, including jellyfish, corals, and sea anemones. Finally, bilaterians are a large clade of animals characterized by bilateral symmetry during embryonic development and constitute over 98% of known animal species. Ctenophores, cnidarians, and bilaterians are three of the five major animal lineages, the other two being Porifera (sponges) and Placozoa.
The hypothesis of monophyly implicates a massive loss of numerous components of the neuronal systems in ctenophores and the complete deprivation of the entire neuronal systems in Placozoa and Porifera[6].
On the other hand, the presence of neural systems in Ctenophora is a significant argument in favor of polygenesis because they are considered to be the earliest branching animal lineage, sister to all metazoans, i.e. all other animals[16, 17]. Furthermore, many molecular components controlling various neuronal functions present in ctenophores are absent in unicellular eukaryotes, which are considered to be sister groups to all animals[6]. Figure 2.3 shows a phylogenetic schematic of the origins of neurons by Moroz [6].
The reasons presented above cause most of the scientific community to be inclined towards the polygenesis hypothesis, and thus a convergent evolution of neurons[6, 16, 18].

Dating back to the Precambrian period, the first hints about the evolution of nervous systems appear in fossils. After the Cambrian explosion (~ 540 million years ago), predation became increasingly more influential and with it the ability of defending oneself from being predated. Hence, in the significantly more complex animal biosphere of the time, natural selection favored the capability of rapid movement and coordination. This progressively more prominent feature facilitated the evolution of a fast-conducting stimulus system like neurons.
Various molecules (voltage-gated channels, molecules that form synaptic structures) specific to neurogenesis or the nervous system were present in all the animal clades prior to the first fossil records. Furthermore, some bacteria present genes homologous to the ones regulating the production of these molecules; thus, these molecules were present even in the prokaryotic/eukaryotic ancestor, which by some is placed as far as 4 billion years ago[18].
Voltage-gated ion channels are a class of transmembrane proteins forming ion channels that are activated by a difference of potential on the cell's membrane. They provide the

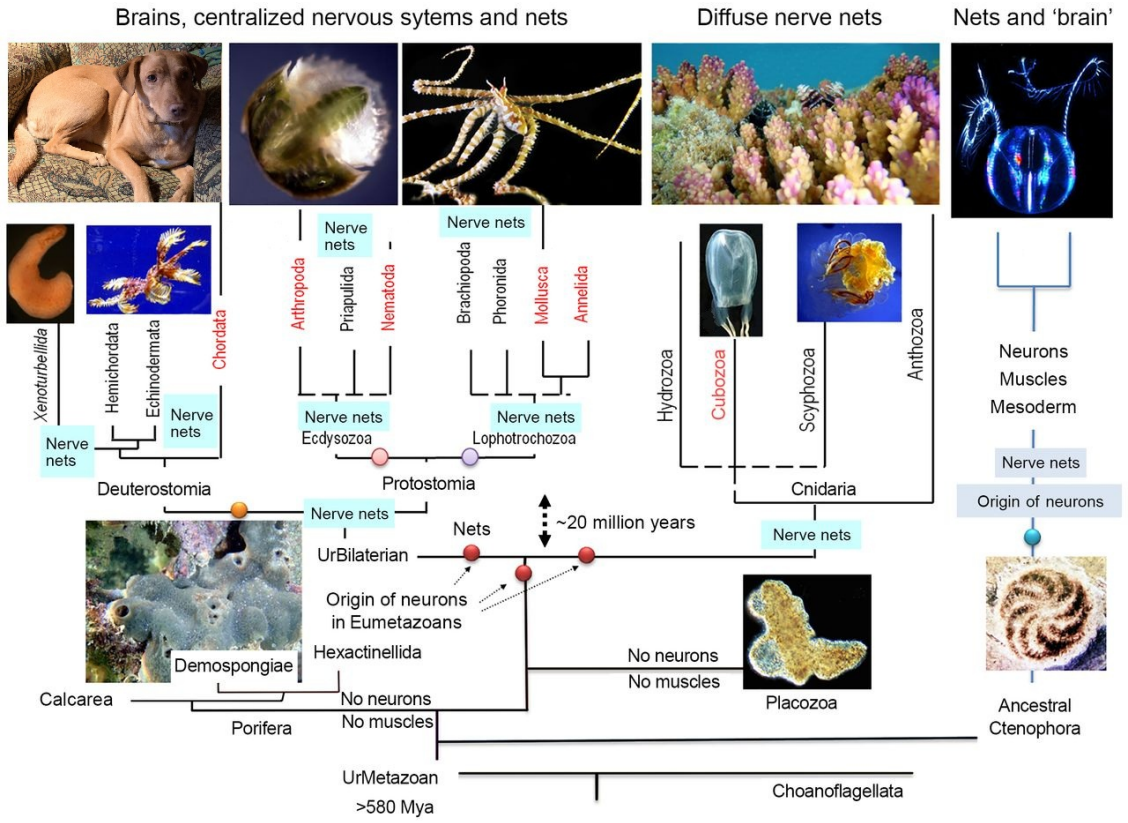

Figure 2.3: A simplified phylogenetic tree of the animal kingdom, combined with chronological information on neuronal evolution. Red phylum names indicate that at least nine independent event of neuronal centralization have occurred. Colored circles indicate possible events of multiple origins of neurons: blue, the origin of neurons in Ctenophores; red, the origin of neurons in Bilateria/Cnidaria clade; it is also possible that some neural populations could originate in bilaterian lineages (yellow, orange and pink circles). Figure adapted from Moroz [6].

base excitation mechanism in various cell types, e.g. in neuronal and muscle tissues[19]. An hypothesis for the function of voltage-gated channels in single-celled organisms is the regulation of intracellular ions and water. Action potentials, which are a series of quick changes in voltage across a cell membrane, are a type of voltage-gated ion channel and the mechanism utilized by neurons to carry information through the neuronal cell's axon[20].

The most accredited theory about neuron evolution places them in as a derivation of epithelial cells[21]. This hypothesis is substantiated by the fact that this type of cells in jellyfish (Cnidaria) generate action potentials, as neurons do, and that neurons in most bilaterians are produced in the ectodermal layer, which produces epithelial cells too. Precambrian animals, such as the placozoan *Trichoplax*, coordinated their behaviour

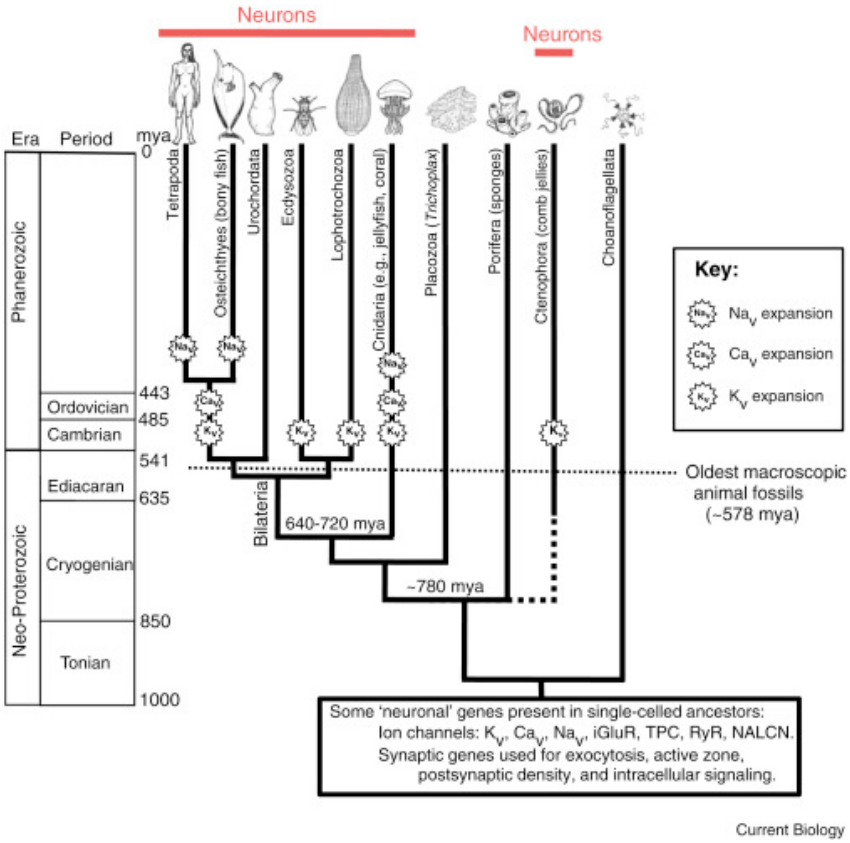

Figure 2.4: Evolutionary tree indicating the diversification of the five major animal clades, indicating some of the major events in the evolution of neurons. The black circles with serrated edges indicate the order of the expansions, mostly through gene duplications, of the genes producing voltage-gated channels selective for potassium (Kv), calcium (Cav), and sodium (Nav) ions. The timing of these events is only approximate. Figure from Kristan [18].

through information carried through epithelial cells. However, especially due to the radiation in animal complexity in the Cambrian explosion, specialization for rapid conduction of some of these epithelial cells yielded survival advantages. In this scenario, these protoneurons would electrically aggregate in nets. Then, once the mechanism to produce chemical synapses developed, neurons started implementing it too, creating links from one neuron to another. Hence, some neurons could now serve as interneurons, transmitting the information from one neuron to another, instead of collecting external stimuli. This development allowed for a significant surge in the complexity of neural nets and thus in processing power[18].

A useful analogy for discussing the functioning of neurons is the one between brains and computers. From Paulin *et al.* [22]:

"Nervous systems originated during the Ediacaran period, as natural computers for predictive statistical inference given event-based sense data."

Pioneering work in mathematical biology by Warren McCulloch and Walter Pitts proved the possibility of formally defining the unit of cognition, i.e., the neuron, in a logical framework. The defining characteristic of the neuron in this model is the threshold response dynamic[23].
The neuron consists in a Boolean system, assuming one of two possible states $\pi_i \in \Sigma = \{0, 1\}$, which correspond, to the inactive and firing state, respectively. In response to input signals from a set of N pre-synaptic units (either other neurons or receptors of external or internal stimuli), the neuron fires if the weighted total contribution of these inputs surpasses a given threshold[5]. Each connection to the neuron i is weighted by a quantity J_{ij}, where j is the pre-synaptic unit. The weights are continuous and can be either positive or negative, accounting for either stimulation or inhibition, respectively. In McCulloch-Pitts model, the integration of pre-synaptic signals determines the state of the neuron i:

$$\pi_i(t+1) = \mathscr{F}\left(\sum_{j=1}^{N} J_{ij}\pi_j(t) - \theta_i\right), \tag{2.1}$$

where the parameter θ_i defines the threshold of neuron i. The activation function $\mathscr{F}(x)$ is a Heaviside function, dealing 1 if x is positive and 0 otherwise. The non-linearity of the activation function enforces the *all-or-none* principle[5].

A crucial result established by McCulloch and Pitts is that networks of such threshold units are capable of implementing any Boolean function. This implies that neural systems, at least in this abstract representation, possess the same computational capabilities as digital logic circuits. This insight laid the foundations for the development of artificial neural networks. In addition to threshold nonlinearities, another important architectural feature emerged: multilayer organisation. Inspired by biological systems such as the visual cortex, layered processing structures have become a central element of modern neural network design [24]. This raises an important question: are threshold units and layered architectures necessary ingredients for computation, or do alternative principles exist? Interestingly, the integration–threshold motif is not restricted to nervous systems. Similar principles appear across a wide range of biological systems, including gene regulatory networks [25, 26, 27, 28], immune systems [29, 30, 31], and collective decision-making in social organisms [32, 33, 34, 35, 36]. It also arises in microbial communities, where quorum sensing provides a clear mapping onto threshold-like decision rules [37]. A notable distinction in many of these systems is that they are

"liquid": their components (e.g., proteins, cells, or individuals) move in space, and interactions are not fixed but dynamically reorganised [38, 39]. Despite this lack of stable connectivity, these systems still implement integration–threshold computations, suggesting a deeper level of universality. Gene regulatory networks provide a particularly well-studied example. In this context, the concentration of transcription factors $[T_i]$ evolves according to

$$\frac{d[T_i]}{dt} = -\lambda_i[T_i] + G\left(\sum_{j=1}^{N} \omega_{ij}[T_j] - \theta_i\right), \tag{2.2}$$

where λ_i is the degradation rate and G is a nonlinear response function. This function arises naturally from the biochemical mechanisms of gene regulation, such as cooperative binding and dimerisation of transcription factors, which inherently produce threshold-like responses [40, 41]. In this case, the weights ω_{ij} capture the influence of different regulators on gene expression.

The use of polarised, threshold-like elements has important consequences for the organisation of biological systems. It enables the emergence of multilayer processing structures, recurrent loops, and memory circuits, all of which are essential for complex computation and learning [42, 43]. Comparative studies across diverse taxa suggest that such architectures may have evolved convergently. For example, both vertebrates and invertebrates exhibit layered neural structures and recurrent connectivity patterns despite independent evolutionary histories [44, 45]. Classical neuroanatomical studies already identified layered organisation in insect and cephalopod visual systems [46, 47]. These observations support the idea that similar computational architectures may arise repeatedly under evolutionary pressures, potentially leading to convergent forms of intelligence and similar architectural principles [48, 49, 50, 51].

2.3 Genetic algorithms

Genetic algorithms (GAs) are a class of stochastic optimization methods inspired by the principles of natural selection and evolutionary dynamics. They operate on a population of candidate solutions (often encoded as strings or vectors), which evolve over successive generations through the application of selection, recombination, and mutation operators. The central idea is that better solutions—those with higher fitness—are more likely to be retained and combined, leading to progressive improvement over time.

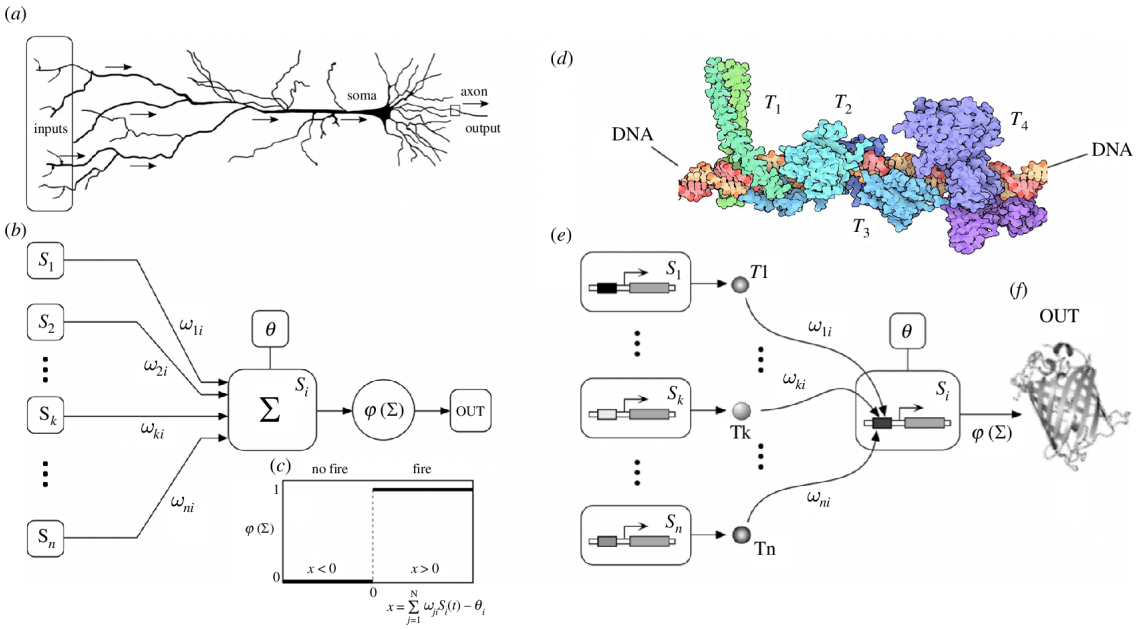

Figure 2.5: Universal transfer functions across biological and computational systems. (a) Schematic representation of a biological neuron, illustrating the integration of multiple inputs at the dendrites, signal processing at the soma, and output transmission through the axon. (b) Abstract formalization of this process as a weighted summation of inputs followed by a nonlinear transfer function, where inputs S_j are combined through weights ω_{ji} and threshold θ. (c) Example of a threshold-like transfer function $\phi(\Sigma)$, highlighting the nonlinear mapping between input and output states (e.g., firing vs. non-firing). (d) Molecular example of gene regulation, where transcription factors interact with DNA to control gene expression. (e) Gene regulatory network analogue of the neural unit, in which regulatory inputs T_j combine through effective interaction strengths to control gene expression via a nonlinear response function. (f) Resulting gene expression output. (Figure from Solé *et al.* [5])

Formally, let $\mathcal{X}$ denote the space of possible solutions (the search space), and let each individual be represented by a genotype $\mathbf{x} \in \mathcal{X}$. A fitness function $F : \mathcal{X} \to \mathbb{R}$ assigns a scalar value to each candidate solution, quantifying its quality with respect to the problem at hand. The objective is typically to find

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in \mathcal{X}} F(\mathbf{x}). \tag{2.3}$$

At generation t, the algorithm maintains a population $\mathcal{P}^{(t)} = \{\mathbf{x}_1^{(t)}, \dots, \mathbf{x}_N^{(t)}\}$ of N individuals. The evolution from $\mathcal{P}^{(t)}$ to $\mathcal{P}^{(t+1)}$ proceeds through three main steps. First, individuals are selected with probability proportional to their fitness, for instance

$$p_i = \frac{F(\mathbf{x}_i^{(t)})}{\sum_{j=1}^{N} F(\mathbf{x}_j^{(t)})}, \tag{2.4}$$

which biases reproduction toward better-performing candidates. Second, selected individuals undergo recombination (or crossover), producing offspring by combining parts of two parent genotypes. Third, mutation introduces random changes in the offspring, ensuring diversity and allowing exploration of new regions of the search space.

A compact representation of the standard genetic algorithm can be written as:

$$\mathscr{P}^{(t+1)} = \mathscr{M} \circ \mathscr{C} \circ \mathscr{S} \left(\mathscr{P}^{(t)} \right), \tag{2.5}$$

where $\mathscr{S}$ denotes selection, $\mathscr{C}$ crossover, and $\mathscr{M}$ mutation. This iterative process is repeated until a stopping criterion is met, such as the convergence of fitness or reaching a maximum number of generations.

As a simple illustrative example, consider maximizing the number of ones in a binary string of length L. Each individual is a vector $\mathbf{x} \in \{0,1\}^L$, and the fitness function is

$$F(\mathbf{x}) = \sum_{i=1}^{L} x_i. \tag{2.6}$$

Starting from a random population, selection favors strings with more ones, crossover combines partial solutions, and mutation flips bits. Over time, the population converges toward the optimal solution $\mathbf{x}^* = (1, 1, \ldots, 1)$.

Genetic algorithms thus provide a flexible and robust framework for solving optimization problems in complex, high-dimensional spaces where gradient-based methods may fail or be inapplicable.

The combination of evolutionary algorithms and neural networks is a promising research field, commonly referred to as '*neuroevolution*'. It consists of the employment of evolutionary algorithms for generating artificial neural networks and their parameters. The advantage provided by neuroevolution over supervised learning algorithms is the wider range of applications. In fact, evolutionary algorithms need only a method for evaluating a network's performance on the task, whereas traditional learning algorithms (such as backpropagation) necessitate large datasets. Furthermore, neuroevolution has been shown to match the performances of gradient descent algorithms in training[53].
In neuroevolution, it is common to assign a computational task to the network and evaluate the network's performance on the task. An example is proposed by Ollé-Vila *et al.* (Figure 2.6), which implemented a neuroevolution algorithm to simulate biological

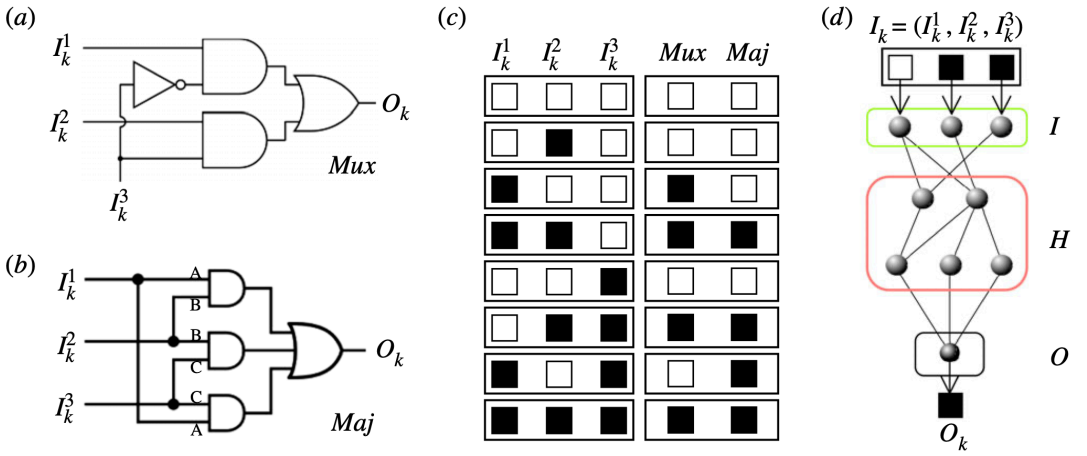

Figure 2.6: Computational tasks and neural implementation. Example of implementation of a Boolean gate with a neural network through neuroevolution: (a) the multiplexer (Mux) and (b) the majority rule (Maj). (c) These are traditionally implemented using logic gates as shown. Each circuit has an associated truth table that fully defines each performed computation. (d) Example of a feedforward neural network architecture formed by input (I), hidden (H), and output (O) layers reproducing a Boolean gate. Figure from Ollé-Vila *et al.*[52].

neural network evolution, similarly to the work presented in this thesis[52].

2.3.1 A simple evolutionary algorithm

Evolutionary algorithms' applications encompass a wide variety of fields, from engineering and agriculture to finance and traffic management[54, 55, 56, 57]. However, most of such usages can be boiled down to minimization problems on an unknown n-dimensional fitness landscape. Therefore, as an example, we consider a most elementary application of evolutionary algorithms: finding the global maximum of a known two-dimensional function, i.e. a simplified fitness landscape.

Definition 2.3.1 (Maximum of a function). Given a generic two-dimensional function

$$\begin{aligned} f \,:\, [A,B]^2 \subseteq \mathbb{R}^2 &\to \mathbb{R} \\ (x,y) &\mapsto f(x,y), \end{aligned} \tag{2.7}$$

where $A, B \in \mathbb{R}$, we can define its maximum (x^*, y^*) as:

$$f^* = f(x^*, y^*) \geq f(x,y), \quad \forall (x,y) \in [A,B]^2$$

where f^* is called the function's maximum value.

The maximum of a function $f : \mathbb{R}^2 \to \mathbb{R}$ can be easily derived from its gradient and Hessian matrix. We start by computing the gradient of f:

$$\nabla_f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \quad ;$$

then, by setting it to zero:

$$\frac{\partial f}{\partial x} = 0 \quad ; \quad \frac{\partial f}{\partial y} = 0$$

and solving the resulting system of equations, we find the potential critical points (minima, maxima, and saddle points) of f. Next, we determine the stability of each critical point through the means of the second partial derivative test, which consists in studying the determinant of the associated Hessian matrix, defined as:

$$\mathbf{H}_f = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} = \begin{pmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{pmatrix} \quad .$$

In particular, given a critical point (a, b) and indicating the determinant of the Hessian matrix on such point as $D(a, b)$:

- if $D(a, b) > 0$ and $f(a, b) > 0$, (a, b) is a local minimum of f,
- if $D(a, b) > 0$ and $f(a, b) < 0$, (a, b) is a local maximum of f,
- if $D(a, b) < 0$, (a, b) is a saddle point of f,
- if $D(a, b) = 0$, the test is inconclusive.

Through such analytical analysis, we are able to find any critical point of a given function f (as defined in Eq. 2.7), and determine whether it is local minimum, maximum or saddle point. However, the second partial derivative test does not offer a way to determine if a given maximum (or minimum) is a global maximum (or minimum), a feature that is often required when studying the critical points of a given landscape.
Therefore, we want to experiment with a different approach and implement a numerical method. It is at this point that evolutionary computation becomes relevant: in fact, we can leverage the power of evolutionary algorithms to search the domain of our two-dimensional function for its global maximum (or minimum).
Normally, the fitness function in an EA is a user-defined function that is hand-crafted

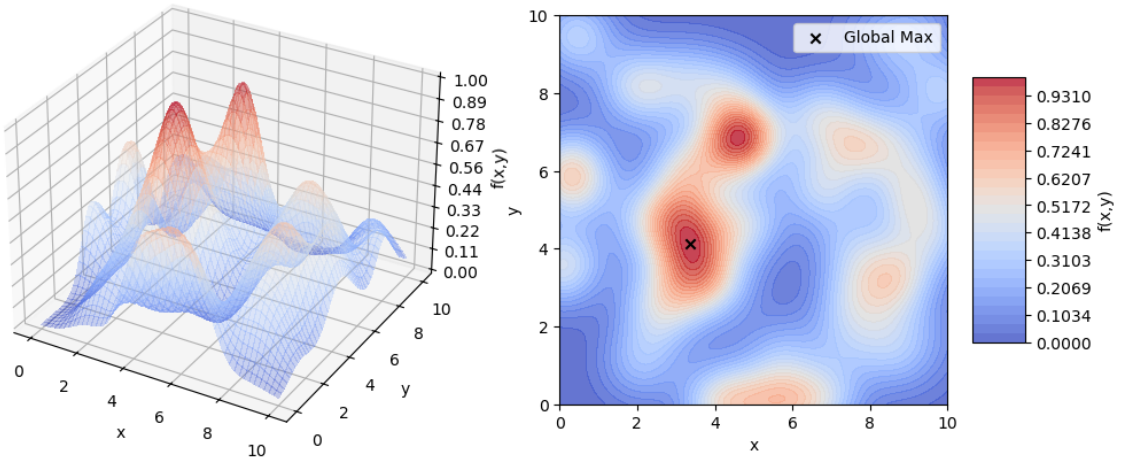

Figure 2.7: Randomly generated 3D landscape and associated isometric top-view heatmap. The 'x' indicates the global maximum, while the heatmap on the right indicates the value $f(x, y)$ of the function at each point in the domain. The z-axis is normalized with the maximum value of f, i.e. f^*

to assist the algorithm in navigating the highly-complex search space towards the problem's solution. In the case of this simplified EA, the search space is fairly simple, and the problem contains only two parameters.

Model

First of all, we generate a random two-dimensional function: this will be the fitness function f of our problem [1]. A visualization of the associated 3D fitness landscape is depicted in Figure 2.7. Secondly, EAs are population-based algorithms; hence, we must start by defining a population of possible solutions:

$$\mathbf{P}^0 = \{(x_i^0,\, y_i^0)\,|x_i^0,\, y_i^0 \in [A, B]\}_{i=[1,M]}\ ,\ \textit{with} \quad x_j^0 \neq x_k^0\ ,\ y_j^0 \neq y_k^0 \quad \forall j \neq k \tag{2.8}$$

at time $t = 0$. On the other hand, for a generic time t, the population of solutions will be indicated as:

$$\mathbf{P}^t = \{(x_i^t,\, y_i^t)\,|x_i^t,\, y_i^t \in [A, B]\}_{i=[1,M]}\ . \tag{2.9}$$

As one can see from the two previous definitions, the population $\mathbf{P}$ is an ensemble of M points (or pairs of coordinates) in the domain D of f, i.e. $\mathbf{D} = [A, B]^2$. Moreover, regarding the initial population $\mathbf{P}^0$, its individuals, i.e. the points of the ensemble, are required

[1] The random two-dimensional function is generated by the sum of 50 two-dimensional Gaussians whose parameters are extracted from uniform distributions, and then it is normalized with its maximum.

to be pairwise distinct. This restriction is lifted for $t > 0$ since there is no reason not to allow a solution to converge onto another.
Now, it is possible to define the algorithm. The intuition behind this EA is to have the population explore the two-dimensional coordinate space $\mathbf{D}$ through evolution and, hopefully, converge to the global maximum. Thus, how do we ensure that the population, after each time step, is moving closer to the global maximum?
The answer of evolutionary computation to this question is to leverage the power of what Darwin called 'natural selection' (or rather a derivation of it). In natural evolution, the individual with a better genetic pedigree to face the challenges posed by the environment will have higher chances, on average, of survival and, thus, of passing , through reproduction, such genetic advantages on to the next generation. In the same way, in evolutionary computation, individuals with a higher fitness value have a higher probability of reproduction, leading the next generation closer, on average, to convergence.
Now, remembering that we chose f as the fitness function, we can associate each individual in $\mathbf{P}^s$ with its fitness:

$$\mathbf{P}^s = \{(x_i^s,\, y_i^s)\}_i \mapsto f(\mathbf{P}^s) = \{f(x_i^s,\, y_i^s)\}_i \quad , \forall s > 0 \,.$$

The population evolved in this EA is placed on a two-dimensional coordinate grid, where each axis is a vector of N evenly spaced values in the interval $[A, B]$. Alternatively, such a grid can be considered a two-dimensional squared lattice of side N, where each site is associated with a coordinate pair.
At this point, we are equipped with all the necessities to define the algorithm.
The EA unfolds over the following steps:

1. **Initial generation**: Generate the initial population of solutions of M individuals, i.e. an ensemble of M pairwise distinct pairs of coordinates $\mathbf{P}^0 = \{(x_i^0,\, y_i^0)\}_i$ (as defined in Eq. 2.8). The population size is defined as $M = \eta N$ where the parameter $\eta \in [0, 1]$ regulates the sparseness of solutions (on the coordinate grid).

2. **Evaluation**: Compute the mean fitness over the population as:

$$\langle f \rangle_{\mathbf{P}^t} = \frac{1}{M} \sum_{i=1}^{M} f(x_i^t\,, y_i^t) \,. \tag{2.10}$$

3. **Selection**: Discard the individuals whose fitness is below average, i.e.

$$f(P_j^t) < \langle f \rangle_{\mathbf{P}^t} \Rightarrow P_j^t \text{ is discarded}\,, \forall j \in [1, M] \,. \tag{2.11}$$

This leaves us with a subpopulation $\mathbf{S}^t$ of $s = M - m$ survivors, i.e.

$$\mathbf{P}^t = \underbrace{\{\mathbf{S}^t\}}_{s \text{ elements}} \times \emptyset^m ,$$

where m is the number of individuals discarded at the previous step.

4. **Parents extraction**: Extract with uniform probability m individuals from the s survivors to be parents. Note that the choice of parents is made '*with replacement*', i.e., the same individual can be chosen to be a parent multiple times in the same extraction step.

5. **Reproduction**: First, define a mutation interval for two coordinates:

$$\begin{aligned} &[-\mu \max_{\mathbf{S}^t} x\,,\ \mu \max_{\mathbf{S}^t} x] \ for\ x \\ &[-\mu \max_{\mathbf{S}^t} y\,,\ \mu \max_{\mathbf{S}^t} y] \ for\ y \end{aligned} \tag{2.12}$$

where μ is the mutation radius, i.e. the magnitude of possible mutation. Then, for each parent p we extract with uniform probability a mutation value for each coordinate, δ_p^x, δ_p^y respectively, from the respective interval defined above. Next, the associated offspring is defined as $c_p = (x_p^t + \delta_p^x\,,\ y^t + \delta_p^y) = (x_c^t\,,\ y_c^t)$. In the case that $x_c^t \notin [A, B]$, then x_c^t is set to the closest boundary (clamping), either A or B (same for y_c^t).

6. **Reconstruction**: Now the population is recomposed for the next iteration:

$$\underbrace{\mathbf{P}^{t+1}}_{N_s \text{ elements}} = \underbrace{\mathbf{S}^t}_{s \text{ elements}} \cup \underbrace{\mathbf{C}^t}_{m \text{ elements}} \tag{2.13}$$

bringing back the population size to its initial value N_s.

7. **Convergence**: Lastly, we check for convergence: the process is said to have converged if $|\langle f \rangle^{t+1} - f^*| \leq \epsilon$, where f^* is the fitness function's maximum value (as defined in Def. 2.3.1) over D, i.e. $f^* = \max_{x,y \in \mathbf{D}} f(x, y)$, and ϵ is a user-defined precision. In the case of convergence, e.g. at time step $t = t^*$, the best approximation of (x^*, y^*), i.e. the maximum of function f, is given by $P_{best}^{t^*} = (x_{best}^{t^*}\,,\ y_{best}^{t^*}) \,|\, f(P_{best}^{t^*}) \geq f(P_i^{t^*})\ \forall i \in [1, M]$. In other words, the individual closest to the global maximum is the one with the highest fitness.
Inversely, if convergence is not reached, we restart from Step 2, utilizing the newly defined population $\mathbf{P^{t+1}}$.

Notes:

- The population size M is conserved between iterations.

N	1000	η	0.2
Max. iterations	200	$[A, B]$	$[0, 10]$
μ	0.05	ϵ	0.01

Table 2.1: Parameters of the EA used in the simulation. From left to right and from top to bottom: N is the length of the two coordinates vector, η regulates the sparseness of solutions ($M = \eta \cdot N$), then the number of generations after which the algorithm is terminated, $[A, B]$ indicates the real-valued interval over which the two coordinate axis of the grid are defined, μ is the mutation radius, and ϵ is the early stop parameter.

- At Step 4, parents could also be extracted using a probability distribution based on their fitness. In that case, extracting the parents '*with replacement*' becomes even more significant since an individual with high fitness, compared to the other survivors, would have a high probability of being chosen as a parent multiple times. This capability is leveraged in the main model discussed in Chapter 4.

- Although the term '*parent*' is used here, one must not assume that this algorithm implements any sort of two-individual reproductive mechanism. Conversely, the process for generating offspring is an asexual one, where the child is a copy of the (single) parent with possible mutations. One could compare it to the biological reproductive mechanism of budding, found, for example, in *Hydra*[58].

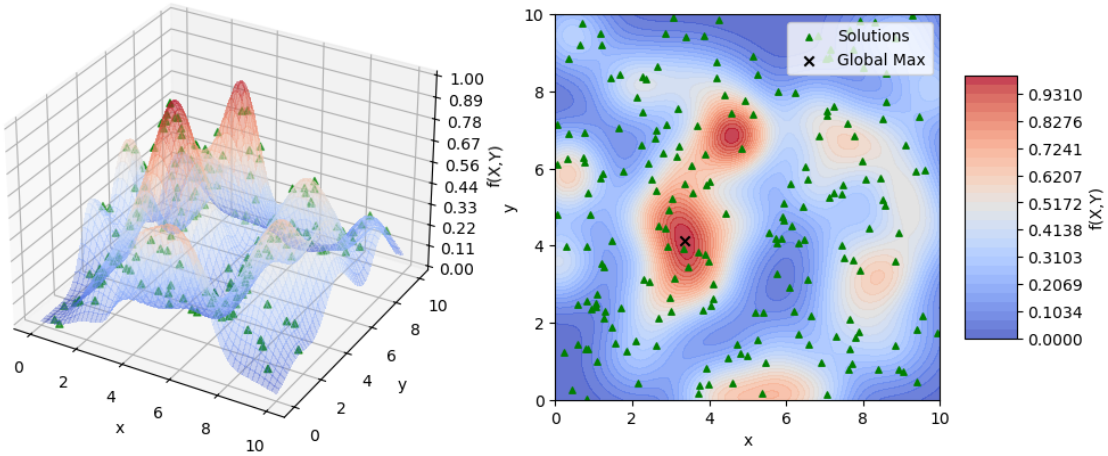

Figure 2.8: Initial random placement of solutions on the landscape. The z-axis is normalized with the maximum value of f, i.e. f^*. The green triangles each indicate a solution's location in the coordinate space.

Results

The EA is complete, and to test it, we can utilize any of the randomly generated two-dimensional functions (f), using the parameters' values from Table 2.1.

As can be seen from Fig. 2.8, we start with a uniform initial placement of solutions, as it should be. Even such a simple EA necessitates just a few generations (or iterations) to drastically restrict the search area in the parameter space. After the surge in fitness of the first generations, the EA continues to refine its accuracy, although slowly, until it converges with a mean accuracy of 0.9901, as reported in Fig. 2.9. Moreover, in this last plot, it is shown how the EA correctly converged to the peak of the global maximum. However, a careful observer will notice that one solution remained stuck in a local maximum's peak. Such occurrences constitute a common problem for simple EAs as this one and search algorithms (e.g., Gradient Descent) in general, especially since the local maximum function value (as can be appreciated from Fig. 2.7) is comparable to the global one, making the differentiation of the two peaks complicated. In more advanced EAs, this problem is avoided through the insertion of random components in the selection and reproduction processes, which will be discussed in depth in Chapter 4.

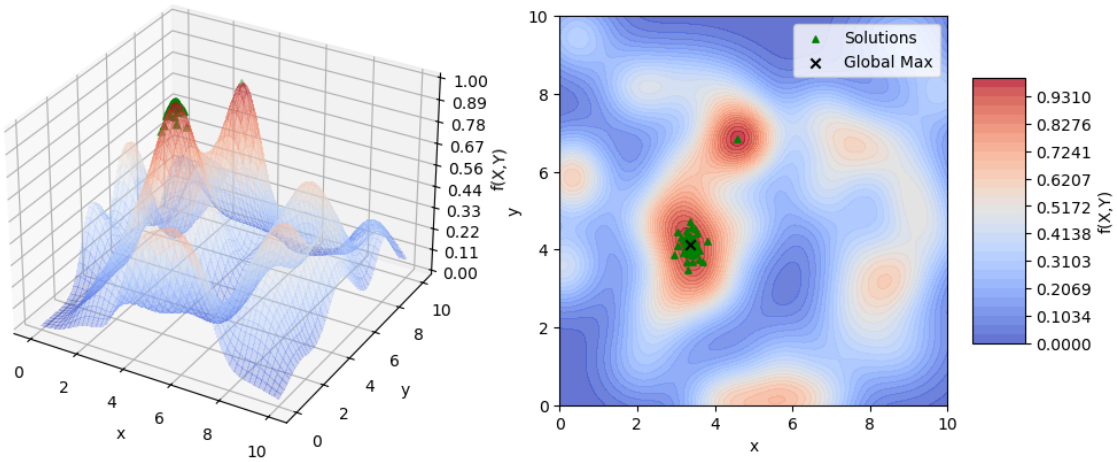

Figure 2.9: Solutions' placement at time of convergence. It can be noticed that one solution is stuck in a local maximum.

3

Perceptron Toy Model

In this chapter, a preliminary study of the main project (Chapter 4) is presented. It consists of the development of a genetic algorithm (GA) to be applied on a population of simple neural networks, with the goal of implementing an XOR logic gate. This task is a commonly used benchmark for testing the rudimentary ability of a machine learning algorithm to learn non-separable data[59]. The main purpose of the work proposed in this chapter is to sample the application of GAs on neural network training and single neuron evolution, while also laying a foundation for the main model to expand on. In particular, this model tests the viability of a GA to substitute classical training methods (e.g. backpropagation[60]) in neural networks.

The XOR is a binary logic gate: it accepts four possible inputs $\{\mathbf{I}\} = \{00, 01, 10, 11\}$ and returns two possible outputs $\{\mathbf{O}\} = \{0, 1\}$. What differentiates any binary logic gate from any other is the output rule, i.e. which input I_i is associated with which output O_j. This rule is commonly represented through one of these two (equivalent) methods: algebraic expression (Definition 3.0.1) or truth table (Table 3.1).

Definition 3.0.1 (XOR logic gate). The XOR logic gate can be defined by the associated algebraic expression:

$$A \oplus B = A \cdot \bar{B} + \bar{A} \cdot B \tag{3.1}$$

where A and B are the first and second inputs, respectively, and $\oplus$ is the symbol for the XOR gate.

The task that neural networks have to solve in this work is to reproduce the XOR logic gate. In other words, a network is considered to have solved the problem if, when given any input $I_i \in \{\mathbf{I}\}$, it returns the corresponding output, as defined in the XOR truth table (Table 3.1).
The work presented in this chapter is an introduction and serves as a preliminary study

Input		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.1: XOR logic gate truth table.

of the main project (exposed in Chapter 4). Therefore, the design introduced is elementary and has room for further optimization. However, this model does not aim to be a computationally effective proposition but rather to test the grounds for the main project.

3.1 Model

Genetic algorithms are population-based, meaning that their computational power originates from the large size of the ensemble of individuals that it evolves. The model developed for this problem employs a population of two-layer[1] feedforward neural networks, where each individual is a possible network structure. The network ensemble is indicated as $\mathbf{P} \subseteq \Omega$, where Ω is the space of all possible networks that satisfy the constraints of the problem.
The fundamental distinction between the method presented and a classical training algorithm for feedforward neural networks is that the weights of the network are not adjusted through a learning algorithm (e.g. backpropagation) but instead the weights' parameter space is explored through the GA's iterations. In practice, this is achieved by making modifications to the weight matrix of the network (later indicated as $\mathbf{J}$) through a reproduction and mutation process, at each iteration. The details behind this procedure and the entire GA are illustrated in depth in Subsection 3.1.2.

Before discussing the composition of the ensemble $\mathbf{P}$, the network object is to be introduced, which is the building block of the population $\mathbf{P}$.

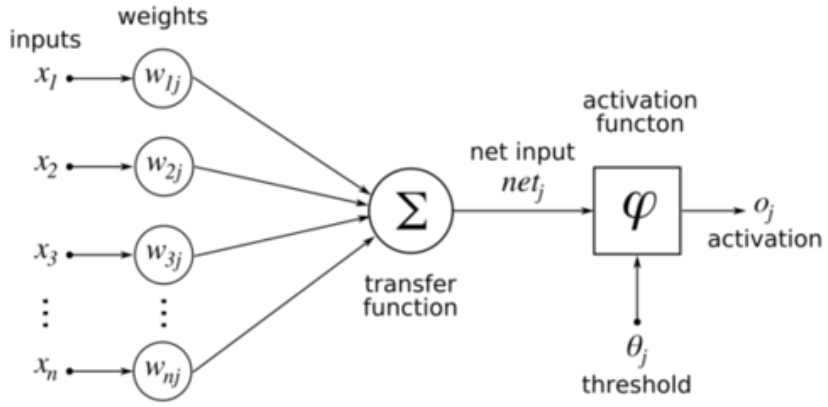

Figure 3.1: Diagram of an artificial neuron.

3.1.1 Network

Each individual P_i in $\mathbf{P}$ is a neural network with three layers: input, hidden, and output layers, in this order. A non-zero number of neurons is placed in each layer, with exactly two input neurons and one output neuron.
The neuron in this model is a classical artificial neuron, which is a cell that receives inputs, processes them by means of an activation function, and returns the result of the computation as output. When placed in a non-input layer of a neural network P_i, the neuron $\pi_j \in P_i$ receives as input the outputs of the neurons of the previous layer and combines them as a linear combination, where the coefficients are the strength (called weights) of the synapses (called links) between a neuron of the previous layer (pre-synaptic) and the receiving neuron π_j (post-synaptic). Figure 3.1 shows a diagram of the structure of an artificial neuron.

The neurons and the synapses between them form a neural network. In this model, each neural network P_i is characterized by some attributes:

- $N_i = 2 + S^i + 1$ neurons: 2 neurons in the input layer, 1 neuron in the output layer and $S^i \in \mathbb{N}$ neurons in the hidden layer (intra neurons).
- A $N_i \times N_i$ connectivity matrix $\mathbf{C}^i$. The element C^i_{kj} expresses the presence (1) or absence (0) of a link from neuron j to neuron k.
- A $N_i \times N_i$ matrix $\mathbf{J}^i$, the weight matrix, containing the weights of each possible link between two neurons of the network. The element J^i_{kj} refers to the synapse from neuron j to neuron k.

[1]It is implied the presence of an input layer prior to the hidden layer, Hence, the structure consists of one input layer, one hidden layer, and one output layer.

- A loss value, i.e. the evaluation on the network of the loss function for the given task.
- A set θ^i of cardinality $|\theta^i| = 2 + S_i + 1 = N_i$ containing the threshold values for the activation function of each neuron.

Moreover, all the neurons of any network in Ω have the same activation unitary step function $f(x;\theta)$, while the threshold θ of such function is characterizing of each neuron.

Now, before the GA is illustrated, each of the attributes listed above will be discussed extensively.
First of all, the architecture: each network is constituted by 3 layers. However, it is customary to simply report the number of hidden layers in the network, which, in the case hereby discussed, is just one. Therefore, each neural network in Ω contains one hidden layer.
Next, the number of neurons in the input and output layers is a direct consequence of the task these neural networks will have to face, i.e. implementing a XOR logic gate. In fact, the XOR is a 2-input logic gate that transforms two binary inputs into a binary output. Therefore, any neural network that has to implement any 2-input logic gate will have 2 neurons in the input layer, one for each binary input to the gate, and one neuron in the output layer to express the binary output of the gate.
On the other hand, there are no pre-imposed constraints on the constitution of the hidden layer, thus rendering the number of neurons S in that layer a candidate for a source of variance in the neural network architecture. Nonetheless, it is necessary to impose a range on the possible values of S, namely $S \in [1, 5] \subset \mathbb{N}$, since more than five neurons in the hidden layer would add unnecessary complexity to the network's architecture[59].

Secondly, the connectivity of the network is discussed. Two matrices, $\mathbf{J}$ and $\mathbf{C}$, encode the links and weights of a network in Ω: $\mathbf{C}$ contains information on whether a given link (or synapse) between two neurons of the network is present, while $\mathbf{J}$ contains the weight (i.e., the strength) of any link. The networks in Ω are forward-directed and lack any self links. Therefore, only links from one neuron in a layer to a neuron in the following layer are possible. This constraint translates to some structural characteristics of the $\mathbf{J}$ and $\mathbf{C}$ matrices: they have only null values (0) on the diagonal (no self links), and they are lower triangular matrices (with, in addition, only zeros on the diagonal), i.e., only the entries below the diagonal are non-zero. This induced structural limitation yields advantages in terms of computational complexity.

Finally, the values that the elements of these two matrices can take are discussed:

- $J_{kj} \in [-1, +1] \subset \mathbb{R}\,,\ \forall\, k, j$. This choice of possible values for the elements of the weight matrix is customary in machine learning since the weight on the link can have up to a double-purpose nature: it may rescale the input from the pre-synaptic neuron ($|J_{kj}| < 1$) and it may transmute that input into either an inhibitory, stimulating, or null (respectively, $J_{kj} < 0\,,\ J_{kj} > 0\,,\ J_{kj} = 0$) effect on the post-synaptic neuron.
- $C_{kj} \in \{0\,, 1\}\,,\ \forall\, k, j$, respectively, absence and presence of the link.

Third, an activation function f is associated to each neuron in each network in Ω. The functional expression (and hence the shape) of the activation function f is common to every neuron in all the networks in Ω and consists in a step function defined as follows:

Definition 3.1.1 (Activation function (toy)). Given a neuron, part of a neural network in the space Ω, its associated activation function f is defined as:

$$f(x) = \begin{cases} 0 & x < \theta \\ 1 & x \geq \theta \end{cases} \tag{3.2}$$

where $\theta \in [-1, +1] \subset \mathbb{R}$ is the value of the threshold attribute associated with the neuron.

The shape of the activation function is biologically inspired, mimicking the *all-or-none* processing mechanism of biological neurons[61].
The activation function f is the same for every neuron, but the threshold parameter θ characterizes each neuron in a network, offering a degree of freedom to explore the parameter space. Thus, each neural network in Ω is, in part, defined by $N = 2 + S + 1$ values of θ, one for each of its neurons. However, it is important to underline that the input neurons, regardless having a threshold attribute θ, they are not characterized by it, since the activation function is not applied on the input neurons. This is a design principle of feedforward neural networks, and it is due to the fact that the sole usage of input neurons is to propagate the inputs given to the network to the neurons in the following layer. Hence, they do not process any data. Nonetheless, they are neurons, and by definition possess a threshold attribute θ.
The parameter θ is defined on the $[-1, +1] \subset \mathbb{R}$ range for the same reasons presented for the elements of the weight matrix and, as for those, it is of capital importance to have the threshold parameter defined on a symmetrical interval around 0, to avoid skewing

the parameter search towards one direction.

Lastly, each network $P_i \in \mathbf{P} \subseteq \Omega$ is characterized by a score, or more precisely, by a value of the loss function. As already seen in Section 2.3, the fitness function is the keystone of any procedure of evolutionary computation, and the GA discussed here is no exception. In fact, the loss function acts on behalf of the fitness function in this case, with the only difference being that the aim of the evolutionary dynamic here is to minimize the loss function, as opposed to maximizing it, as is done with fitness functions. The loss function characterizing this GA is defined as:

Definition 3.1.2 (Loss function (toy)). Given a neural network P_i part of the population $\mathbf{P} \subseteq \Omega$, the loss function is defined as:

$$L(P_i) = \frac{1}{4} \sum_{\{I\}} \left[\alpha \left(O^T - O \right)_I^2 + S_i \left(\sum_{l,m}^{N_i} C_{lm}^i \right) \right] \tag{3.3}$$

where the sum is over the 4 possible inputs to the network $\{I\} = \{00, 01, 10, 11\}$, α is a weight for the first term of the sum, O^T is the theoretical output of a XOR logic gate given input I, while O is the actual output of network P_i given input I. S_i is the number of intra neurons in the network P_i hidden layer, $N_i = 2 + S_i + 1$ is the total number of neurons in the network P_i , and $\mathbf{C}^i$ is the network's connectivity matrix.

As it has been shown in this section, each network in $P_i \in \Omega$ is defined by a set of attributes, namely $S_i, \theta^i, \mathbf{C}^i, \mathbf{J}^i$, which completely determines its structure and output for a given input. Therefore, the possible network space Ω can be expressed as:

$$\{\Omega\} = \{\{S_i\}, \{\theta^i\}, \{\mathbf{C}^i\}, \{\mathbf{J}^i\}\}_i$$

3.1.2 Genetic algorithm

The network and population structure have been introduced, and it is now time to delve into the intricacies of the genetic algorithm in play here.
The GA unfolds over the following steps, which are briefly described here:

1. **Initialization**: Generate the initial network population $\mathbf{P} \subseteq \Omega$ of size M and initialize each individual (or solution).

2. **Evaluation**: Compute the loss value (following Definition 3.1.2) for each individual. The best individuals have the lowest loss.

3. **Selection**: Discard the solutions whose loss is *above* the mean population loss $\langle L \rangle_{\mathbf{P}} = \frac{1}{M} \sum_i^M L(P_i)$.
4. **Extraction of parents**: Randomly choose $m = \rho \cdot s$ parents from the s survivors, where $\rho \in [0, 1] \subset \mathbb{R}$ is a parameter referred to as the mutation ratio, which regulates the ratio of new individuals between generations by mutation and random generation. Note that the selection of parents is done '*with replacement*', i.e. the same individual can be chosen to be a parent multiple times.
5. **Mutation**: For each parent, generate an offspring by mutating the parameters $S, \theta, \mathbf{C}$ and $\mathbf{J}$.
6. **Random generation**: Generate the remaining individuals to reconstitute the original population size through random generation.
7. **Convergence**: Check for convergence of the method. If converged, stop the execution; otherwise, restart from Step 2 with the newly defined population.

The steps just described are the general structure of the GA, and now they will be individually discussed in depth.

Initialization: To generate the initial population $\mathbf{P} \subseteq \Omega$, $|\mathbf{P}| = M$ individual neural networks are instantiated and then initialized. The initialization method utilizes randomly generated values for the parameters (with the exception of the functional definition of the activation function f, as previously mentioned).
In particular, for each network $P_i \in \mathbf{P}$:

- $S_i \sim \mathcal{U}(2, 5)$.
- $\theta_i \sim \mathcal{U}(-1, +1)$, for each neuron $\pi_i \in P_i$.
- C^i_{kj} is set to either 0 or 1 with uniform probability, $\forall\, j, k \in [1, N_i]$.
- $J^i_{kj} \sim \mathcal{U}(-1, +1)\,,\ \forall\, j, k \in [1, N_i]$.

Naturally, in the initialization of J and C, the elements on the diagonal and above it are set to zero, and they will remain zero for the entire execution of the algorithm since they are not relevant to the problem analyzed here.
Lastly, the loss value of each network $P_i \in \mathbf{P}$ is initialized to zero.

Evaluation: For the initial population, the loss is computed right after initialization, while for the following generations, it is computed at the start of a new iteration. In

both cases, the loss value is derived following Definition3.1.2 and is calculated individually on each network $P_i \in \mathbf{P}$. The construction of the loss function, the reasons behind its form, and its computation in detail will be discussed in the following Subsection 3.1.3.

Selection: This is the step that causes evolution to move forward or, in other words, that tries to lower the mean loss of the network population $\mathbf{P}$. As it has been done in the simple evolutionary algorithm presented in Subsection 2.3.1, the selection procedure consists in cutting off from the population the individuals who are reporting below-average performance (i.e. above average loss values), in order to then substitute them with new solutions. As in Subsection 2.3.1, the cutoff threshold is provided by the mean population loss, defined as:

$$\langle L \rangle_{\mathbf{P}} = \frac{1}{M} \sum_{i}^{M} L(P_i) \,. \tag{3.4}$$

Extraction of parents: In this step, $m = \rho \cdot s$ parents are chosen from the s individuals who have survived the selection process. As in Subsection 2.3.1, the m parents are extracted from the survivors with uniform probability, and this random selection process is carried out '*with replacement*', i.e. the same survivor can be chosen multiple times to be a parent. The mutation ratio parameter $\rho \in [0, 1] \subset \mathbb{R}$ expresses the fraction of new individuals that will be generated by mutation of (some) old individuals (parents). The remaining part of the missing individuals (to recompose the initial population size M) will be created by random generation (as will be discussed further later).

Mutation: In this phase, m new individuals are generated through asexual reproduction with mutation. This process is more complex than the one presented in Subsection 2.3.1, but it follows the same logic, and, as that one, it is inspired by the asexual biological reproductive dynamic of budding (e.g. fungiid corals [62]). An offspring P_j is generated as a copy of the associated parent $P_i \in \mathbf{P}$, i.e. having the same defining attributes except for the connectivity matrix, namely $S_j = S_i, \theta^j = \theta^i, \mathbf{J}^j = \mathbf{J}^i$. Successively, mutations are applied to the offspring P_j, i.e. its defining attributes are modified. The connectivity matrix $\mathbf{C}$ is not inherited, as doing so would result in a more challenging convergence. Instead, in any offspring, a new connectivity matrix is initialized (following the same procedure seen in Step 1 of the GA) and then mutated.
The mutation process for each attribute is worth discussing individually:

- S: mutation value χ_S sampled from $\{-\mu_S, 0, +\mu_S\}$ with respective probabilities $\{0.2, 0.6, 0.2\}$, where μ_S is the mutation radius for the number of intra neurons

S. The mutation value on S is not sampled with a uniform probability since it can dramatically modify the structure of the network and thus its loss value. Therefore, the mutation on such attributes is skewed towards a null mutation. Otherwise, there would be a high risk that the algorithm would not be able to draw a qualitatively stable path towards convergence.

- θ: mutation value χ_θ sampled from a Gaussian distribution $\mathcal{N}(0, \mu_\theta)$, where the parameter μ_θ is the mutation radius.
- $\mathbf{C}$: since the elements of matrix $\mathbf{C}$ are binaries, i.e. $C_{lm} \in \{0, 1\}$, $\forall\, l, m$, here the mutation is defined in a slightly different manner. In fact, the mutation consists of a flip of the element value (bit flip), for each element of $\mathbf{C}$. Such mutation (or flip) is applied to each element with a probability μ_C. Obviously, a mutation is applied only to the matrix elements below the diagonal since the others are irrelevant to the problem and are set constant to zero.
- $\mathbf{J}$: mutation value $\chi_\mathbf{J}$ sampled from a Gaussian distribution $\mathcal{N}(0, \mu_J)$, where the parameter μ_J is the mutation radius. Naturally, as for the connectivity matrix, a mutation is applied only to the relevant elements of the matrix, i.e. the ones below the diagonal.

With the exception of the connectivity matrix, once all the mutation magnitudes have been calculated, it is time to apply them to the defining attributes $S_j, \theta^j, \mathbf{J}^j$. These modifications are implemented by summing the previously computed mutation magnitudes with the inherited attributes.
In other words, given an offspring network $P_j \in \mathbf{P}$ during the mutation step, its mutated attributes are computed as follows:

$$S'_j = S_j + \chi_{S_j} \tag{3.5}$$

$$S'_j = \theta^j + \chi_{\theta^j} \tag{3.6}$$

$$\mathbf{J}'^j = \mathbf{J}^j + \chi_{\mathbf{J}^j} \tag{3.7}$$

Lastly, once these alterations are implemented, all the modified elements of $S_j, \theta^j, \mathbf{J}^j$ are clamped to the respective range enforced by the corresponding definitions.
To reiterate, the mutation on the connectivity matrix attribute $\mathbf{C}_j$ does not involve the summation step just described, since the mutation on that parameter involves only a bit flip on each element of the matrix with a probability $\mu_\mathbf{C}$.
A couple of further notes regarding the mutation step:

- The choice of the ρ parameter is highly influential on the evolution dynamics, as it regulates the equilibrium between exploration of the search space and advancement on the generational path indicated by the fittest individuals.

- If S is mutated, then all the other defining attributes have their sizes modified, since $|\theta| = N$, $\mathbf{J}, \mathbf{C}$ are $N \times N$ matrices, and $N = 2 + S + 1$. Therefore, in the mutation step, after S has been successfully modified, two cases are possible:

  A. $S_j < S_i$: the number of intra neurons, and with it the sizes of the three attributes, is shrunk by $\Delta S = |S_i - S_j|$. Hence, the rows and columns in $\mathbf{J}^j, \mathbf{C}^j$ and the elements in θ^j , corresponding to the 'lost' intra neurons, are eliminated.

  B. $S_j > S_i$: the number of intra neurons, and with it the sizes of the three attributes, is increased by $\Delta S = |S_i - S_j|$. Hence, in $\mathbf{J}^j, \mathbf{C}^j$ and θ^j, the corresponding rows, columns, and elements (respectively) are added. These newly added components are then initialized, following the same procedure and rules applied in Step 1 of the GA. Naturally, if, in the same mutation step, $\mathbf{J}^j, \mathbf{C}^j$, or θ^j are mutated, the components just added due to the mutation on S_j are also mutated.

- Extracting mutations from a normal distribution centered in zero (as it is done for θ and $\mathbf{J}$) is a common practice in evolutionary computation, as it allows for high flexibility. This is generally preferred, as it results in an isotropic exploration of the search space.

Random generation: In the previous step, $m = \rho \cdot s$ new individuals (offspring) were generated. At the end of this phase, the population of networks $\mathbf{P}$ must be back to its original size, i.e. M. The new population will be formed by up to three groups of individuals based on their origin:

1. $s \leq M$ individuals (survivors) who passed the selection phase, i.e. whose loss is below the old population's $\mathbf{P}$ mean loss $\langle L \rangle_{\mathbf{P}}$.

2. $m = \rho \cdot s \leq s$ solutions (offspring) created by reproduction from some of the survivors.

3. $r = M - s - m$ networks from random generation.

In this phase, this third group of r individuals is created.
The random generation process is analogous to the initialization process, and the new solutions are formed following the same procedure.
This additional stage may seem negligible; however, it is of capital importance to the convergence of the method, as it enables the GA to explore the search space, avoiding

being stuck in a local minimum of the loss function.

Convergence: At the end of each iteration of the GA, convergence is verified. The first condition for convergence is that the task has been successfully reproduced. Once that is verified, the second (and last) check is performed by inspecting the number of offspring generated in the mutation stage. In particular, if that quantity is less than or equal to the parameter ϵ, then an early stopping (of the GA) is triggered. This method is preferred to one based on a threshold on the loss value since it does not require specifying such a limit, which could prove to be a difficult challenge and easily lead to errors. On the contrary, by implying the number of offspring and, as a direct consequence, the number of discarded individuals in the selection process, it is a more organic procedure, triggering once the population has converged to a similar loss value.

3.1.3 Loss function

The loss function is the beating heart of any genetic algorithm, as it paves the way for the algorithm to move towards convergence. In this GA, the loss is defined as already stated in Definition 3.1.2, which is reiterated here. For a given neural network $P_i \in \mathbf{P} \subseteq \Omega$:

$$L(P_i) = \frac{1}{4} \sum_{\{I\}} \left[\alpha \left(O^T - O \right)_I^2 + S_i \left(\sum_{l,m}^{N_i} C_{lm}^i \right) \right]$$

As one would notice, the above function is comprised of two terms:

1. $\alpha \left(O^T - O \right)_I^2$: This part computes the squared difference (or distance) between the expected output O^T and the actual output O, for a given input I. It leads the GA towards the expected outcome by favoring the solutions that perform better at the given task (in this case, reproducing the XOR logic gate). A coefficient, α, is present in front of this term: it is added to weight the influence of this term on the whole loss, and its value is inferred from testing. This term, aside from α, is used as the first check for convergence.

2. $S_i \left(\sum_{l,m}^{N_i} C_{lm}^i \right)$: This is a cost term. In particular, it affects the connectivity of the network: the more links that are present, the higher this part is. Moreover, it favors smaller hidden layers by multiplying the term by S. The reasoning behind this term is that simpler and sparser networks are preferred, since, in a biological setting, each link or neuron comes with an energy cost.

Lastly, the sum of these two terms is averaged over the whole input set, which in this case is $\{I\} = \{00, 01, 10, 11\}$ and consists of 4 inputs.

At this point, to compute the loss value for a given network P_i, the only missing piece is to extrapolate the output of the neural network from the inputs. This procedure is carried out through what is known in machine learning as *feedforward*: it consists of setting the values of the input neurons to the input values and then computing the states of the subsequent neurons. This process is regulated by a modified version of the classical expression used in feedforward neural networks:

Definition 3.1.3 (Learning rule (toy)). Given a neuron $\pi_i \in P$:

$$\pi_i(t+1) = f\left(\sum_j^N C_{ij} J_{ij} \pi_j(t)\right) \tag{3.8}$$

where $\mathbf{C}, \mathbf{J}$ are the connectivity and weight matrices, respectively, of network P, f is the activation function, and t is the time (or iteration) step.

In the classical definition of feedforward neural networks, the connectivity matrix is not present since every link (from a neuron in one layer to another neuron in the layer just before or after) is supposed to always exist, and to the sum is added a scalar term (bias) associated with the post-synaptic neuron π_i.
After the sum of all the contributions from the neurons is computed, the activation function f is applied to the result, giving the final state of the post-synaptic neuron in the next time step. The activation function utilized in the model described in this chapter is the one defined in Definition 3.1.1. As can be seen from that definition, f is a step function with threshold θ. In a given neural network $P_i \in \mathbf{P} \subseteq \Omega$, the threshold value for each neuron $\pi_j \in P_i$ is none other than the estimate of the associated attribute $\theta_j \in \theta_i$, which has been discussed above. Therefore, through the GA, the threshold of the activation function for each neuron $\pi_j \in P_i$ is mutated, altering the output of P_i and thus its loss.

3.2 Results

The parameters utilized in the simulations, whose results are presented in this section, can be inspected in Table 3.2.

Two main simulations have been carried out, and they differ by the value given to a structural boolean option: whether to allow direct links from the input layer to the output one (skip connections) or not. Indeed, in feedforward neural networks, neurons can only be linked to neurons in the layers just before and after theirs. Therefore, only

S range	$[1,5] \subset \mathbb{N}$	μ_S	1
θ range	$[-1,+1] \subset \mathbb{R}$	μ_θ	0.1
J range	$[-1,+1] \subset \mathbb{R}$	μ_J	0.1
M	400	μ_C	0.8
ϵ	1	ρ	0.8
α	100	N. iterations	1000

Table 3.2: Values of the parameters used in the simulations. From left to right and from top to bottom: range of possible number of neurons in the hidden layer, mutation radius of parameter S, range of possible values for the threshold parameter θ for any neuron, mutation radius of parameter θ, range of possible values for the weights, mutation radius of the weights, population size, mutation probability of each element of the connectivity $\mathbf{C}$, maximum number of offspring to be generated in the reproduction step for triggering an early stop, fraction of survivors becoming parents, weight coefficient on the score term in the loss function, maximum number of generations before the execution is halted.

one of the possible options would lead to an architecture in line with the definition of feedforward neural network. These two cases will from now on be labeled as 'Case A' and 'Case B' (or 'Type A architecture' and 'Type B architecture'), respectively.

After convergence, the networks have correctly reproduced the XOR logic gate (as defined in Table 3.1) with an accuracy over the population $\mathbf{P}$ of 99%. The final population's (i.e. $\mathbf{P}(t^*)$, where t^* is the time of convergence) performances on the task are shown in Figures 3.6,3.7, where it can be verified that, in both cases, the task has been

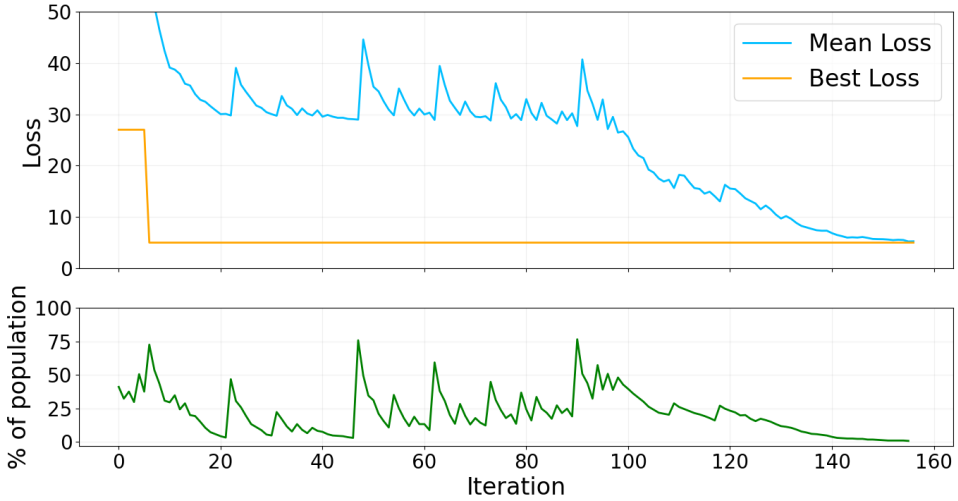

Figure 3.2: Loss and selection percentage evolution for type-A architectures. Top: loss value evolution as in the best individual's loss and in mean loss over the population. Bottom: percentage of population discarded in the selection step at each iteration.

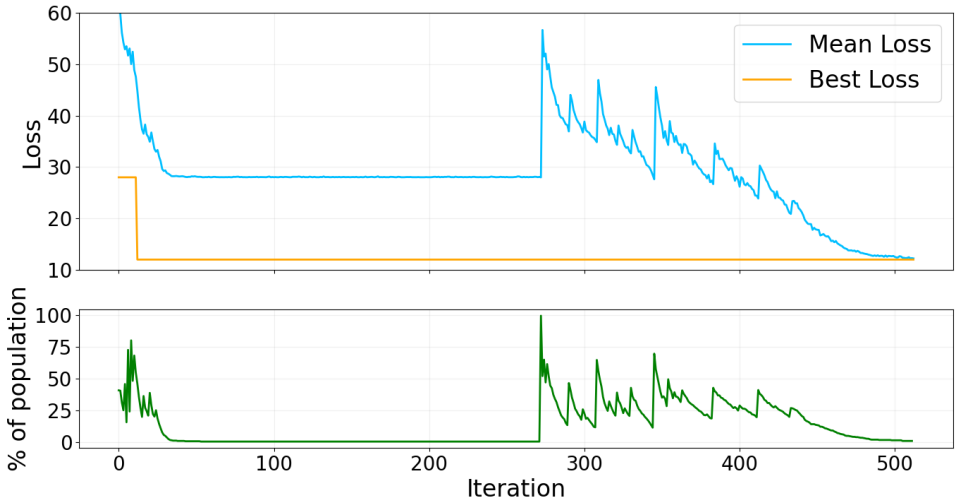

Figure 3.3: Loss and selection percentage evolution for type-B architectures. Top: loss value evolution as in the best individual's loss and in mean loss over the population. Bottom: percentage of population discarded in the selection step at each iteration.

completed successfully. However, the loss value at convergence differs in the two instances. In fact, case A converged at $\langle L(t^*)\rangle_{\mathbf{P}} = L_{best} = 4.7$, while case B converged at $\langle L(t^*)\rangle_{\mathbf{P}} = L_{best} = 12.0$, as seen in the upper graphs of Figures 3.2,3.3.
On another note, it is clear from Figures 3.2,3.3 (top), when looking at the best individual from **P**, that the problem was solved in a considerably lower time (i.e number of iterations). Thus, the greater part of the simulation time was spent on 'moving' the whole population (i.e. the mean quantities) to the best individual. This is the standard behaviour for these types of GAs, where individuals who have already solved the task will always advance to the next generation (see the rules of the selection step in Subsection 3.1.2).

Next, in Figures 3.8,3.9 (bottom), the mean number of neurons in the hidden layer over the population ($\langle S\rangle_{\mathbf{P}}$) is shown at each iteration for cases A and B, respectively. The first thing to notice is that the GA tries to minimize this quantity, as it is clear from its evolution in the first iterations for both cases. This evidence is not a surprise since it comes directly from the expression of the loss function applied in this problem. In fact, as can be checked in Equation 3.3, the secondary objective of the GA is to minimize the connectivity expense in the network, and naturally, this translates into an advantageous selection towards networks with a lower S value. Nonetheless, in Figure 3.9 (bottom), it is evident how $\langle S\rangle_{\mathbf{P}}$ stabilizes at $S = 2$, thus not minimizing the quantity to its lowest possible value, $S = 1$. However, networks of type B are still successfully completing the

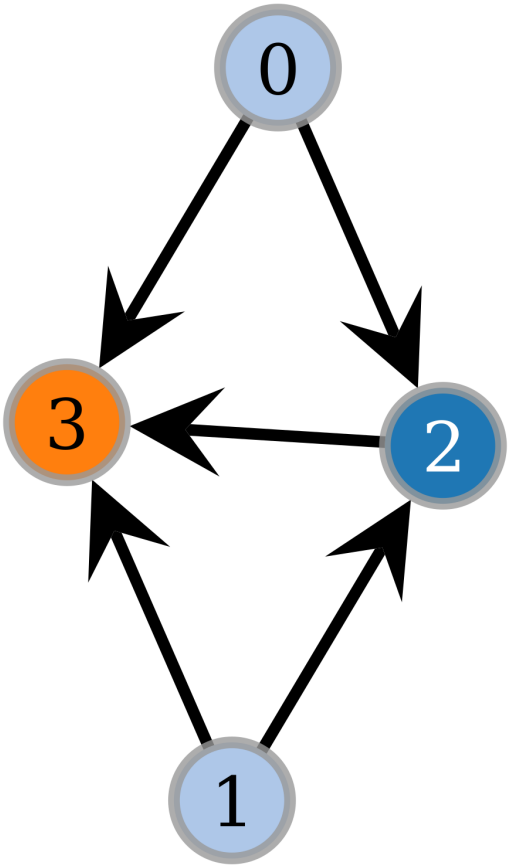

Figure 3.4: Network graph of the best solution for case A. 0-1: input neurons, 2: hidden neuron, 3: output neuron.

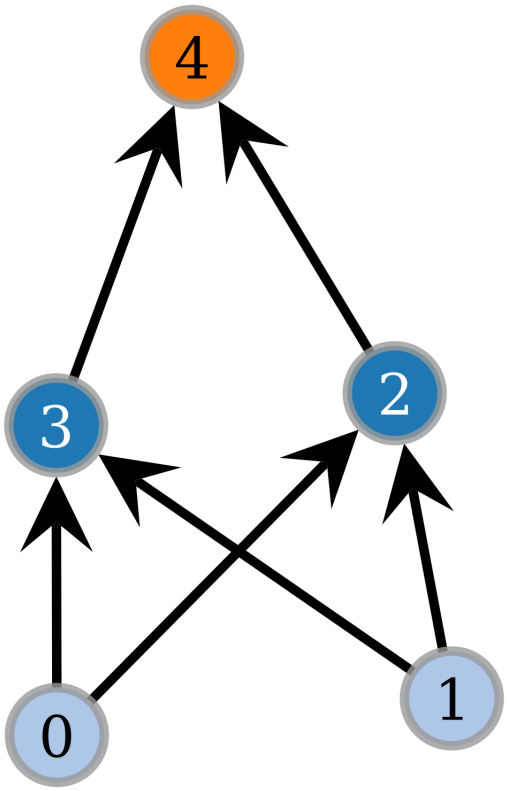

Figure 3.5: Network graph of the best solution for case B. 0-1: input neurons, 2-3: hidden neurons, 4: output neuron.

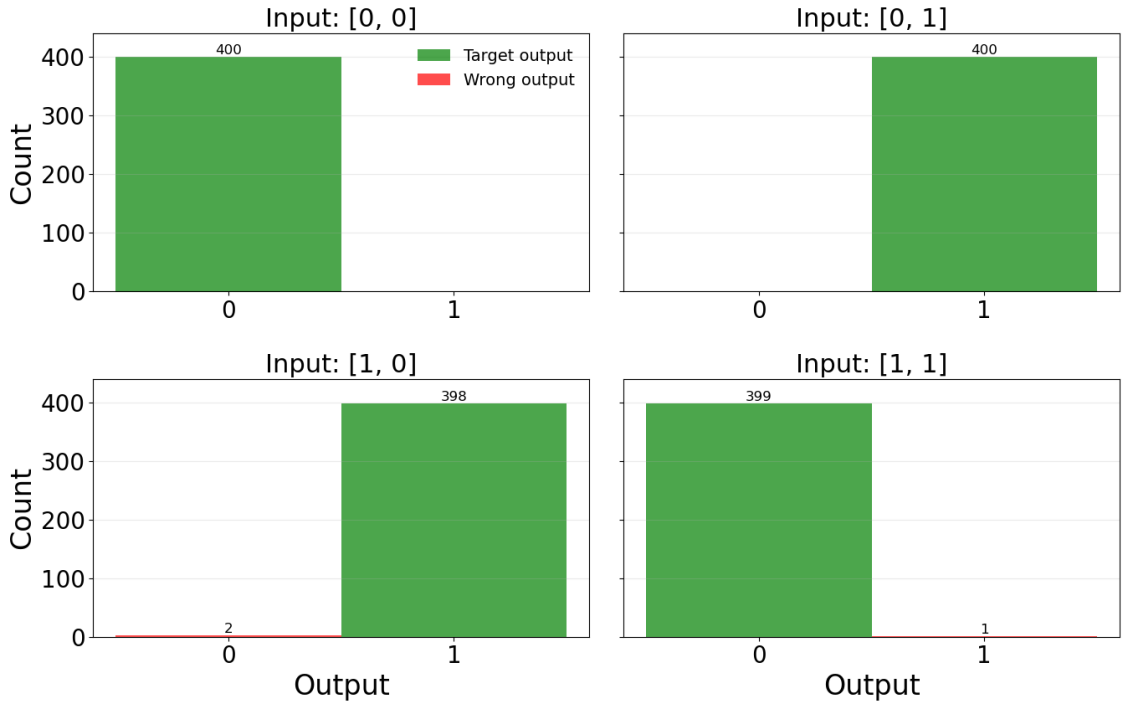

Figure 3.6: Output distribution given each input - Case A

task with practically the same accuracy as type A networks. Moreover, it is apparent from the evolution of the loss of the best individual in Figure 3.3 (top) that the GA found the solution to the task in the first iterations, meaning that it was able to solve it easily even with the limitation on direct input-output links.

All of this evidence points to the fact that without direct input-output links, the network needs at least two neurons in the hidden layer to be able to reproduce the XOR logic gate (Figure 3.5). This coincides with a fundamental notion of machine learning, established from the early discoveries in the field of neural networks: single-layer perceptrons are able to learn only linearly separable data. Hence, since type B networks with a single hidden neuron are single-layer perceptrons, they are unable to learn the XOR gate, which is not a linearly separable problem. On the other hand, multilayer perceptrons, a class to which type B networks with at least two hidden neurons belong, are notoriously able to learn non-separable data, with the XOR problem being their benchmark[59].

Instead, type A networks are not multiclass perceptrons, and they are not even feedforward neural networks, since in those architectures, direct input-output (or skip) connections are not allowed. However, other kinds of neural networks implement these sorts of connections, such as Cascade-Correlation neural networks[63] or Residual neu-

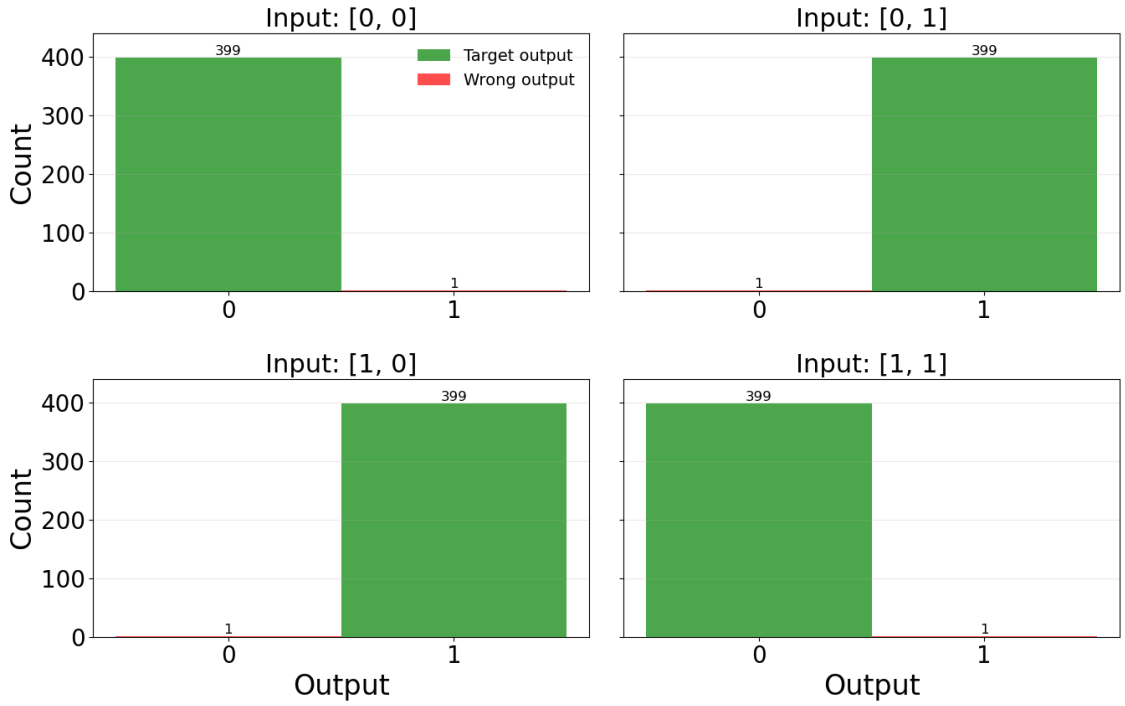

Figure 3.7: Output distribution given each input - Case B

ral networks[64]. Even the father of the perceptron, Frank Rosenblatt, described neural networks with skip connections[65]. Overall, it is widely recognized that architectures such as the one proposed as a solution for case A (Figure 3.4) can learn a non-separable problem such as the XOR.

The attentive reader will have noticed the simultaneous spike in the evolution of all the graphed quantities for both architecture types. In particular, this happens around iteration $t \simeq 130$ and $t \simeq 50$ for cases A and B, respectively. In order to fully understand the event that unfolded here, it is instructive to focus on the selection process. In particular, in Figures 3.2, 3.3 (bottom), the fraction of elements discarded in the selection process is reported at each iteration. Vividly, it can be seen how the spike in the mean loss evolution aligns with the one in the bottom graph. What this clue tells it is that these spikes are due to the discovery of a new solution that out-performed the previous population, causing an evolutionary shift. This could be geometrically visualized as finding an unexplored higher maximum of the fitness function in the simple EA described in Subsection 2.3.1 (see Figure 2.7). This behaviour is particularly interesting because it mimics what happens in biological evolution when a mutant of a species presents a crushing selective advantage over its peers.

Moreover, Figures 3.8, 3.9 explicitly show that these spikes locate the moment when

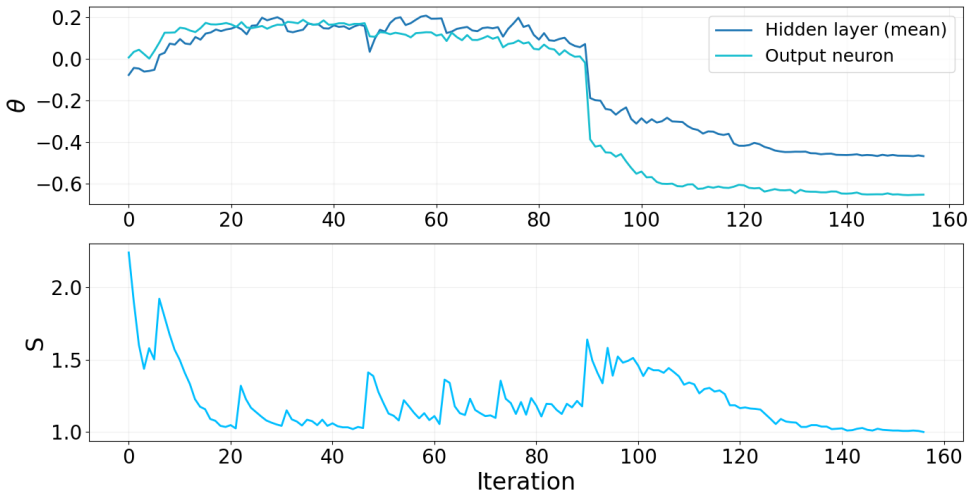

Figure 3.8: Evolution of the mean of attributes θ and S for case A. Top: mean value over the population $\mathbf{P}$ of the threshold parameter θ for neurons in different positions. For the hidden layer neurons, the mean over $\mathbf{P}$ is computed on the mean value of the attribute θ over all the neurons in the hidden layer. Bottom: mean number of neurons in the hidden layer over the population $\mathbf{P}$.

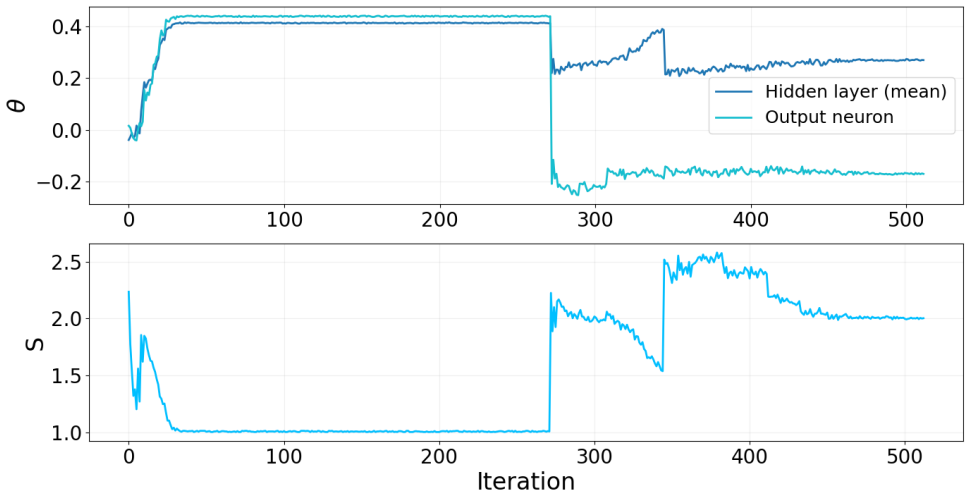

Figure 3.9: Evolution of the mean of attributes θ and S for case B. Top: mean value over the population $\mathbf{P}$ of the threshold parameter θ for neurons in different positions. For the hidden layer neurons, the mean over $\mathbf{P}$ is computed on the mean value of the attribute θ over all the neurons in the hidden layer. Bottom: mean number of neurons in the hidden layer over the population $\mathbf{P}$.

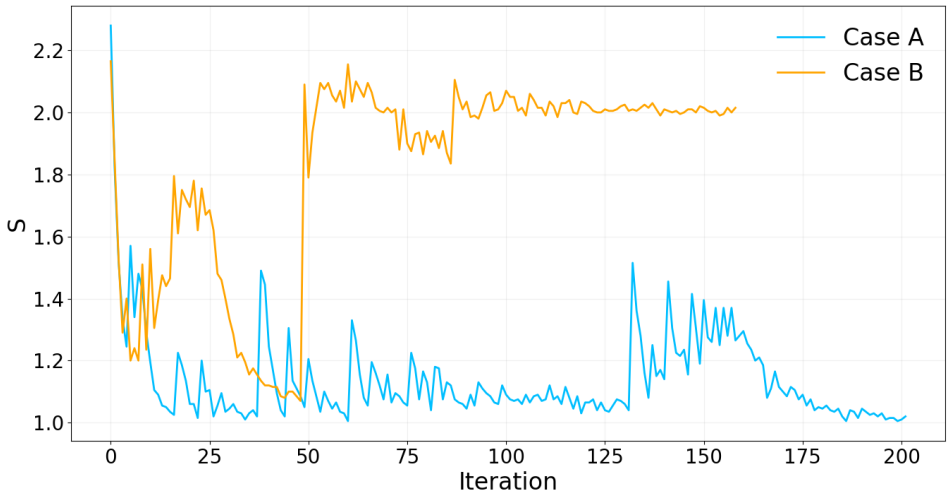

Figure 3.10: Evolution of the mean value over the population of attribute S for both cases. S determines the number of neurons in the hidden layer.

the GA found the final solution to the task. In fact, in Figure 3.9 can be seen how the spike at $t \simeq 50$ corresponds to the shift from most of the population having one hidden neuron (configuration which, as seen before, is not able to complete the task) to two hidden neurons, i.e. the correct value for the parameter. Lastly, as further confirmation of this hypothesis, after the spike, various quantities stabilized to what will then be the final value.

To conclude, this model and the provided results confirm that GAs are a viable alternative for neural networks' training to classical methods, such as backpropagation. This result is preliminary to the main model, which will be discussed in the next Chapter.

4

Genetic algorithm for neuron evolution

The capability of storing and accessing information becomes necessary to organisms when dealing with a complex, ever-changing external environment. On the other hand, processing signals and learning are powerful abilities to reduce the uncertainty of the surroundings[5]. Cognitive systems offer a substrate for developing these skills, resulting in a crushing advantage in the arena of natural selection. Furthermore, the advent of multicellular life at the Cambrian explosion prioritized goal-oriented movement and enhanced environmental perception[22].
Most cognitive systems present a mechanism to convert analogue signals into digital responses. The necessity for signal distinction and the higher susceptibility to error of analogue computation might constitute the two major constraints on the evolution of cognition[5, 66].
The evolutionary emergence of neurons and neural systems offered multicellular organisms a significant advancement over diffusion-based signal transmission[67]. In neural units, processing decisions are made following a standard design principle: input integration is weighted and processed through threshold-mediated response dynamics[5]. Is this a universal design principle?.

In this chapter, the main model developed in this project is presented. It consists of systems of excitable cells organized in a population and evolved through a genetic algorithm. These excitable cells are proxies for the ancestors of neurons (protoneurons) and act as a clean slate for evolution to draw on. The cells are embedded in a squared lattice, which serves only to fix their spatial disposition, and can evolve connections between each other. Various characteristic parameters of the cells and of the network are evolved by means of the genetic algorithm, with the objective of efficiently solving

a given task.
The purpose of the network of cells is twofold: in fact, it acts both as a representative of a (proto)neural system and as a substrate for the cells to evolve on. Hence, the cells evolve at the same time as components of the network and as independent possible models of a fundamental unit of cognition.

4.1 Model

The evolving population in this final iteration is a system of excitable cells embedded in a square lattice, which can develop connections between each other. Hence, the individuals of this population are networks of cells, as in Chapter 3, albeit they do not present any form of organization in layers, as is customary in neural networks. The system of cells is indicated as $\mathbf{P} \subseteq \Omega$, where Ω is the space of all possible networks satisfying the problem's constraints.
When compared to the model presented in the previous chapter, this version aims to employ a form as general as possible. The objective behind this decision is to have the least possible number of imposed assumptions on the system in order to apply minimal influence on the outcomes. Nevertheless, some constraints and assumptions are present in this first iteration of the model, since it has been preferred to prioritize the completion of a functional algorithm, with the promise to remove as many limitations as possible once the final version is completed.

4.1.1 Network of excitable cells

The neuron in this model, at the most general level, is an artificial excitable cell that takes some input, processes it by means of a transfer function, and returns some output. Furthermore, each individual in the population $\mathbf{P}$ is a system of these excitable cells, with each one occupying a site of a fully-occupied square lattice of side L. Hence, each network contains $N = L^2$ cells. The cells cannot move on the lattice, i.e. they remain in the same site for as long as they are present on the lattice. Besides, at the start of each GA's iteration, there is exactly one cell in each site of the lattice. In other words, the lattice acts as a system of coordinates in two dimensions, with a finite interval on which each coordinate is defined, i.e. the side of the lattice. The single purpose of the lattice embedding is to define a measure of distance between the cells, which will be used in computing a cost term in the loss function.
Moreover, two sites on the lattice are selected to be the locations of the two input cells

and one other site for the output cell. These positions are parameters of the simulation and common to every individual in **P**.
The learning mechanism, i.e., the method through which the cell integrates information from other cells, is (again) a modification of the classical learning rule of feedforward neural networks, and it is expressed in Definition 4.1.1.

Definition 4.1.1. Given a neuron π_l part of a network $P \in \mathbf{P}$:

$$\pi_l(t+1) = \Phi_l\left(\sum_k^N \mathbf{C}_{lk}\,\mathbf{J}_{lk}\,\pi_k(t) + \mathbf{B}_l\right) \tag{4.1}$$

where $\mathbf{C}, \mathbf{J}, \mathbf{B}$ are the connectivity, weight, and bias matrices, respectively, of network P, Φ_l is the transfer function of neuron π_l and t is the time (or iteration) step.

Each cell can receive inputs from any other cell in the network, regardless of the spatial distance between them. However, as will be described in more depth later, longer links, i.e. direct connections between two cells that are further apart on the lattice, are discouraged. Furthermore, self links, i.e. direct connections from one cell to itself, are not permitted. This design choice stems from the fact that self loops are difficult to balance and risk invalidating the learning capabilities of a network.
The network object is defined by several attributes:

- A $N \times N$ connectivity matrix $\mathbf{C}$. The element $C_{kj} \in \{0\,,1\}$ expresses the presence (1) or absence (0) of a link from cell j to cell k.
- A $N \times N$ matrix $\mathbf{J}$, the weight matrix, containing the weights of each possible link between two cells of the network. The element $J_{kj} \in [-1,+1] \subset \mathbb{R}$ refers to the direct connection from cells j to cell k.
- A set $\mathbf{B}$ of N real numbers, containing the bias value for each cell.
- A loss value, i.e. the evaluation on the network of the loss function for the given task (see Subsection 4.1.3).
- A set $\mathbf{T}$ of N real numbers which stores the value of the gain parameter of the transfer function of each cell in the network.

Each of these attributes is reviewed in depth later in the section.
Therefore, a crucial difference from the model presented in Chapter 3 is that there are no assumptions about the structure of the network, since it is a system of cells that can freely evolve connections between one another, without any preferred direction for the

transmission of information.

Before delving into the details of the higher-order structures of this model, the attributes of the network object listed above are explained thoroughly.
Starting with the connectivity of the network, which is regulated by two matrices: the connectivity matrix **C**, and the weight matrix **J**, which encode the links and weights, respectively, between any two cells in the network. **C** stores the information on whether the direct connection between any two cells is present, while each element in **J** determines the strength of the corresponding connection. The structure of these matrices is determined by constraints imposed by the problem: self links are not permitted, thus the diagonal of both matrices is fixed at zero. Then, in this version, there is not a preferred direction for the information to flow through the network, meaning that the elements of the two matrices above and below the diagonal are relevant and **C**, **C** are not symmetric matrices. Lastly, between any two cells, at most two links can be present, one per direction; in other words, there can be at most one link from cell π_j to cell π_k, for any two cells in any network in Ω.
Secondly, the bias matrix: a bias value $B \in \mathbb{R}$ is associated with each cell of the network and stored in the **B** matrix. The bias term is a fundamental element in any neural network, because it allows the model to cover a wider range of inputs. In this study, this addition is due to its use as a proxy for baseline modulations of inputs, which is a universally present phenomenon in Nature, caused by a wide range of influences, from environmental shifts to arousal[68].
Finally, each network $P \in \mathbf{P} \subseteq \Omega$ is characterized by a value of the loss function. As in the previous model, the aim of the GA is to minimize this function over the population of networks **P**: Hence, the value of the loss function computed on a network acts as a score for positioning that system of cell in the population. The definition and details of the loss function applied in this model are discussed further in Subsection 4.1.3.
In conclusion, each element P_i in the space of possible networks Ω can be defined by the attributes described above:

$$\{\Omega\} = \{\{\mathbf{C}^i\}, \{\mathbf{J}^i\}, \{\mathbf{B}^i\}, \{\mathbf{T}^i\}\}_i \ .$$

4.1.2 Genetic algorithm

The evolution of the population $\mathbf{P} \subseteq \Omega$ of network is carried out by means of a genetic algorithm, whose purpose is to mimic evolution as closely as possible. In this section,

that GA is discussed.
As it is common when describing algorithms, the structure of the GA is explained by illustrating each one of its steps:

1. **Initialization**: Generate the initial network population $\mathbf{P} \subseteq \Omega$ of size M and initialize it.
2. **Evaluation**: Compute the loss value (following Definition 4.1.3) of each individual. The best individuals have the lowest loss.
3. **Parents selection**: Extract $m/2$ pairs of parents with replacement from the entire population, where $m = \rho \cdot M$. The parameter $\rho \in [0, 1] \subset \mathbb{R}$ is the reproduction ratio. Each solution's selection probability is given by the softmax of its negative loss.
4. **Reproduction**: Each pair of parents generates two children by crossover recombination and/or mutation.
5. **Mutation**: Mutate each offspring's attributes with the associated probability.
6. **Reconstruction**: Generate the remaining individuals to reconstitute the original population size through random generation (without mutation) and elitism.
7. **Convergence**: Check for convergence of the method. If converged, stop the execution; otherwise, restart from Step 2 with the newly defined population.

At this point, each step of the algorithm presented above is discussed individually.

Initialization: To create the initial population $\mathbf{P} \subseteq \Omega$, $|\mathbf{P}| = M$ systems of excitable cells embedded in a fully occupied square lattice are instantiated, and their attributes are initialized. In particular, for each network $P \in \mathbf{P}$, its attributes $\mathbf{J}, \mathbf{B}, \mathbf{T}$ are initialized at zero, while for the connectivity matrix, each element is set to zero with probability $1-\zeta$ and to one with probability ζ, with the exception of the diagonal, which is fixed at zero.

Evaluation:The value of the loss function on each individual $P \in \mathbf{P}$ is computed, following Definition 4.1.3. The loss function is discussed further in Subsection 4.1.3.

Parents Selection: The aim of this step is to better the performance of the population, or, in other words, to move the ensemble of solutions closer to the global minimum of the loss function. In this last iteration, the selection process is more complex than what was employed in the previous cases. In fact, a more advanced selection mechanism translates directly into a stronger GA, which can avoid getting stuck in local minima of

the search space, as was occurring with the perceptron toy model (see Figure 3.3).
In this phase, $m = \lfloor \rho \cdot M \rfloor$ individuals organized in pairs are extracted from the population to be parents. When selecting a pair of networks from **P** as parents, any solution in **P** can be extracted, with the only rule being that the two parents must not coincide. Hence, the selection process is performed with replacement: any individual in **P** can be chosen to be a parent multiple times.
Additionally, each solution's selection probability is defined as the softmax of its negative loss:

$$p(P) = \frac{e^{-\tau \mathscr{L}(P)}}{\sum_k^M e^{-\tau \mathscr{L}(P)}} \tag{4.2}$$

where $\mathscr{L}(P)$ is the loss function computed on network P and $1/\tau$ is the temperature parameter.
This is the most substantial difference from the previous GA's selection process. In fact, here the probability of becoming a parent and thus reproduction is a function of the loss, and in particular, individuals with lower loss will have a higher likelihood of selection. Employing a loss-based selection probability is a major improvement in the GA's power since it allows the process to be flexible, improving convergence time and substantially lowering the probability of the simulation remaining stuck in local minima. Furthermore, this method shows a higher resemblance to its biological counterpart, i.e., Darwin's notorious *survival of the fittest*.

Reproduction: From any given pair of parents, two children are generated. The possible reproduction mechanisms are twofold:

1. **Crossover**: This method emulates crossover recombination in sexual reproduction in Nature. Crossover recombination is employed only for generating the gain parameter of the children. The **T** set of child A is formed by the union of the first k elements of the **T** set of parent 1 and the $k+1$ to M elements of the **T** set of parent 2, and vice versa for child B. In other words,

$$\mathbf{T}^A = \mathbf{T}^1_{1:k} \cup \mathbf{T}^2_{k+1:M}$$
$$\mathbf{T}^B = \mathbf{T}^2_{1:k} \cup \mathbf{T}^1_{k+1:M}$$

   where $\mathbf{T}^X$ is the gain parameter's set for child/parent X. The crossover index k, i.e., the position in the set where the chiasma is taking place, is extracted with a uniform distribution for each pair of parents. The other attributes of a child are a copy of one parent's and the other parent for the other child.

2. **Budding**: Each attribute of one child is a copy of the corresponding attribute of

one parent and of the other parent for the other child.

For each pair of parents, reproduction employs crossover recombination with a probability of γ (crossover probability). The reasoning behind a partial application of crossover is that it greatly increases the instability of the system, by adding heavy randomness to the reproduction process. For this argument, crossover is employed with parsimony and govern by a user-defined parameter. Especially in the case of the connectivity and weight matrices, crossover renders the learning of the task considerably more difficult.

Mutation: After reproduction, each offspring's attributes are mutated. The mutation procedure is analogous to the one of the perceptron toy model in Chapter 3. In particular, each attribute is mutated with the following reasoning:

- **C**: Each element has a probability of bit flip μ_C, meaning that it can be mutated to the opposite value ($0 \to 1/1 \to 0$) with probability μ_C. Naturally, the mutation does not involve the diagonal elements of **C** since they are fixed at zero by definition.

- **J**: The mutation on each element is extracted from a normal distribution centered in zero with standard deviation μ_J. The values are then summed to the corresponding value.
$$J'_{kl} = J_{kl} + \mathcal{N}(0, \mu_J)\,, \ \forall k, l$$
where **J′** is the mutated version of the weight matrix **J**. As for the connectivity matrix, the mutation does not involve the diagonal elements of **J** since they are fixed at zero by definition

- **B**: Same rule as for **J**:
$$B'_k = B_k + \mathcal{N}(0, \mu_B)\,, \ \forall k$$
where **B′** is the mutated version of the bias set **B**.

- **T**: Analogous to the two cases above:
$$T'_k = T_k + \mathcal{N}(0, \mu_T \cdot L)\,, \ \forall k$$
where **T′** is the mutated version of the gain parameter set **T** and L is the side length of the square lattice [1].

Reconstruction: In this phase, the remaining $M-m$ individuals to reconstruct the population size M are generated. The individuals are created either by random generation

[1] Recall that the lattice is fully occupied, thus $L = \sqrt{N}$, where N is the number of cells in the lattice.

or through an elitist process. Elitism is a common procedure in evolutionary computation, in which the algorithm retains some number of the best-performing individuals at each generation. These networks do not go through the process of mutation[69].
The step can unfold in two ways, depending on whether a satisfactory solution has been found:

(A) **Random generation + elitism**: $\lfloor (M-m)v \rfloor$ elitists are selected and will be part of the new population. These individuals are the $\lfloor (M-m)v \rfloor$ best solutions in the population in terms of loss. The remaining networks needed to reconstitute the initial population size M are randomly generated: this process is identical to the initialization procedure. These newly created solutions do not undergo any mutation phase.

(B) **Elitism**: All the $M-m$ remaining individuals to reconstruct the population are elitist, i.e. are the best $M-m$ solutions of the previous generation in terms of loss.

There are two conditions to be met for the random generation to be halted permanently: if the best loss (i.e. the lowest) is lower than or equal to the user-defined reconstruction parameter κ, and that the full form of the loss function has already been deployed [2]. If these two requirements are verified, the random generation is stopped, meaning that the population size is reconstructed through elitism (B) for the remaining part of the simulation. Otherwise, random generation and elitism are employed in tandem (A). This dual system is implemented because the purpose of random generation is to explore the search space, and such capability is not necessary once the system has found a satisfactory solution. Rather, once the GA has located an adequate area for convergence, random generation causes it to slow down.
At the end of this phase, the new population has been formed.

Convergence: At the end of each iteration of the GA, convergence is verified. Due to the different selection process in this algorithm compared to the perceptron toy model, the condition for convergence has to be based on a threshold on the loss of the population. In particular, the algorithm is considered to have converged if and only if at least $\epsilon_2 \cdot M$ individuals have loss lower than ϵ_1. The parameters ϵ_1, ϵ_2 are user-defined, and their values are tuned through trial and error.
An attentive reader would notice that this convergence criterion is fundamentally different from the ones employed in the other two algorithms. In fact, while convergence was previously based on the performance of the whole population by inspecting the

[2] This second condition is explained in detail in Subsection 4.1.3

mean loss, it is now not required for the GA to lead the entire population to the optimal solution, but rather only a (user-defined) fraction of it. Additionally, this inherently distinct criterion is preferred because the convergence of the entire population is not necessary and would greatly hinder performance.

4.1.3 Loss function

The loss (or fitness) function is the core of any model of evolutionary computation since it is what leads the population to convergence through evolution. In fact, the Darwinian principle of the *survival of the fittest* is translated in numerical simulation through evolutionary computation, and the quantifier of such fitness is the loss (or fitness) function of the model.
Before defining the loss function, it is needed to clarify on the measure of distance between cells on the square lattice. Each site on the lattice is uniquely defined by two coordinates, x, y, and the position of each cell coincides with the coordinates values of the site in which it is located. Consequently, the distance between two cells is the distance between their respective lattice sites. Then, the distance on the lattice is defined as:

Definition 4.1.2. Given two sites $A, B \in E$ on a square lattice, uniquely identified by a pair of coordinates, (A_x, A_y) and (B_x, B_y), respectively, the distance between these two sites is defined as:

$$\begin{aligned} d \,:\, E &\longrightarrow \mathbb{R}^+ \\ (A, B) &\longmapsto \left\| (A_x - B_x, A_y - Bl_y) \right\|^2 , \end{aligned} \tag{4.3}$$

where $\|\cdot\|$ stands for the Euclidean norm.

The loss function of this model follows the same reasoning as the one of the perceptron toy model (Subsection 3.1.3) and is defined as:

Definition 4.1.3. Given a system of excitable cells P part of population $\mathbf{P} \subseteq \Omega$, the loss function is defined as:

$$\mathscr{L}(P) = \frac{1}{n_I} \sum_{\{I\}}^{n_I} (O^T - O)_I^2 + \left(\frac{\sum_{k,l}^{M} J_{kl}\, d(k,l)}{L(L-1)} \right)^2 \tag{4.4}$$

where n_I is the number of possible inputs to the network, i.e. $|\{I\}| = n_I$, where $\{I\}$ is the set of possible inputs. Then, O^T is the target output, while O is the actual output of the

network, both given input I. Next, J_{kl} is the weight from cell l to cell k and $d(k,l)$ is the distance between cell k and cell l, as in Definition 4.1.2. Lastly, $L = \sqrt{N}$ is the side length of the square lattice onto which the network of cells is embedded.

The loss function defined above is composed of two terms:

1. $\frac{1}{n_I} \sum_{\{I\}}^{n_I} (O^T - O)_I^2$: This part is a score term and it computes the squared difference (or distance) between the expected output O^T and the actual output O, for a given input I. It leads the GA towards the expected outcome by favoring the solutions that perform better for the given task.

2. $\left(\frac{\sum_{k,l}^{M} J_{kl}\, d(k,l)}{L(L-1)} \right)^2$: This is a cost term, and it affects the connectivity of the network, favoring simpler, more diffused, and localized connective structures. In particular, it impairs stronger as well as lengthier links between cells by including in the loss the weight of the link (J_{kl}) and the distance ($d(k,l)$) between the two cells, respectively. The reasoning behind this term is that simpler networks are to be preferred since, in a biological setting, each link comes with an energy cost, and that cost is higher for stronger and/or longer connections. Specifically, in this model, the cost term and its design are crucial since cost-effectiveness is a golden rule in natural selection, and this study aims to reproduce a biological setting as closely as possible.

A couple of additional notes on the form of the loss function:

- Although the form of the loss function utilized in the simulations is the one above (Definition 4.1.3), the cost term is introduced only after at least one individual in the population has a satisfactory loss. In particular, this means that the cost term is permanently added to the loss functional expression once the best score (i.e. the lowest) in the population is lower than a user-defined parameter η. This mechanism is implemented to help the GA at the start of the simulation and to have it prioritize solving the task rather than simplifying the connectivity. Indeed, if the entire form of the loss function is employed from the beginning, it is likely that the GA will converge towards architectures that have too few links and thus are unable to solve the task.

- The connectivity matrix **C** is not included in the cost term since every time **J** is modified, it is multiplied by **C** in order to set non-existing links to zero. On the other hand, the bias set **B** is not included because its role in this model is to be a proxy for features external to the evolution (natural conditions, arousal, etc...) and thus serve a strictly functional role. In other words, biases cannot be optimized by evolution.

The last piece of information needed about the loss function and its evaluation is how to compute the output of a network $P \in \mathbf{P}$. This process is similar to the feedforward mechanism seen for the perceptron toy model (Subsection 3.1.3) and it consists of 2+1 steps:

1. Set the value of the input cells with one of the possible inputs $\{I\}$ of the task.
2. Propagate the inputs through the network following the learning rule from Definition 4.1.1, here repeated:

$$\pi_l(t+1) = \Phi_l\left(\sum_k^N \mathbf{C}_{lk}\,\mathbf{J}_{lk}\,\pi_k(t) + \mathbf{B}_l\right)$$

These two steps are repeated α times in order to compute the output of the network at a state closer to the steady state. In particular, at each iteration, the input cells' values are set with the inputs, while for the other cells (including the output), their values are retained from the previous iteration. Then, at the end of the last iteration:

3. Extract the output of the network as the value of the output neuron.

4.1.4 Transfer function

A crucial point of divergence from this model and the perceptron toy model is in the transfer function. In fact, the transfer function employed in this case is fundamentally different from the step function defined in Definition 3.1.1:

Definition 4.1.4 (Transfer function). Given a cell π_l part of network $P \in \mathbf{P} \subseteq \Omega$, its transfer function is defined as:

$$\begin{aligned} &\Phi_l : \mathbb{R} \longrightarrow \mathbb{R} \\ &\Phi_l(x) \longmapsto \frac{\lambda}{1+e^{-gx}} - \frac{\lambda}{2} \end{aligned} \tag{4.5}$$

where λ is the amplitude of the response and g, the *gain* parameter, which determines the slope of the curve. The second term $(\frac{\lambda}{2})$ ensures that the curve is symmetric with respect to the input axes.

The gain parameter is initialized to zero for every cell, so that the initial transfer function returns zero for any input value. Then, through the GA, this parameter is mutated, and the transfer function takes shape.
This form of the transfer function is essentially distinct from that of the previous model since its output is not binary. In fact, although for high absolute values of the gain parameter g it resembles a step function, the image of the transfer function employed in this model is the interval $\mathcal{I}m(\Phi) = [-\lambda/2, +\lambda/2]$. Hence, the output of the network, coinciding with the transfer function computed on the final value of the output neuron, can take any value in $\mathcal{I}m(\Phi)$, not only $\{0, 1\}$.
The reasoning behind this design principle is the same as that which is the basis of this entire model: the systems of cells should be placed under as few constraints as possible. In other words, ensuring that the search space is maximally vast is a design priority, and the only limitations imposed on it must come directly from the biological and evolutionary arena.
Furthermore, this freedom for the network to produce real-valued outputs is a necessary condition for the model to be able to answer one of the two main points of this study's question, i.e., whether non-threshold response dynamics are evolutionarily feasible in a cell dedicated to the function of cognition.

4.2 Results

As it was for the perceptron toy model, the performance of a network in this model is evaluated on a binary task, and that task is, again, to reproduce the XOR logic gate. The reasons behind this choice are the same as those presented for the case in Chapter 3. As a reminder, the XOR is a binary logic gate, which translates two inputs $(00, 01, 10, 11)$ into an output $(0, 1)$. It follows the rule presented in Definition 3.0.1 and repeated in the associated truth Table 3.1. However, here it is employed a slightly modified version of the XOR gate, whose truth table is shown in Table 4.1. The logic of the gate is obviously analogous to the 'classical' XOR gate, but the negative value (0) has been replaced with -1. The argument for this decision is simply that, because of the image of the transfer function and the continuous cells' value domain, it is necessary to have a task which is symmetric around zero, in order to avoid skewing the results towards one direction.

After this brief introduction, some results are presented. The parameter values utilized for the simulations producing these results are shown in Table 4.2.

Input		Output
A	B	$A \oplus B$
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Table 4.1: Modified XOR logic gate truth table. This is the task employed for the main model.

After convergence was reached, the GA was able to lead most of the population to solving the task, i.e. reproducing the XOR gate, with an accuracy over the population $\mathbf{P}$ of 76% on the final population $\mathbf{P}(t^*)$, where t^* is the time of convergence. The distribution of the outputs of the final population is shown in Figure 4.1. These distributions highlight an interesting feature of this model: the output of a network is not binary (i.e. either 0 or 1), but rather a real number in $\mathcal{I}m(\Phi) = [-\lambda/2, +\lambda/2]$. Additionally, as expected, most of the instances of classification error occurring consist of a network associating an input to its opposite label (i.e. $\{1, -1\} \to -1$ instead of the correct $+1$). This is anticipated since the task consists of a non-separable binary problem. However, the percentage of false positives (5.9%) is an acceptable error rate, given the premises of the study.

L	4	μ_C	0.1
λ	2	μ_J	0.05
τ	500	μ_B	0.05
ρ	0.4	μ_T	0.05
ν	0.4	α	4
γ	0.6	ϵ_1	0.1
κ	0.01	ϵ_2	0.9
η	0.05	M	200
N. of iterations		1000	

Table 4.2: Values of the parameters used in the simulations. From left to right and from top to bottom: side length of the squared lattice, mutation (bit flip) probability of each element of the connectivity matrix $\mathbf{C}$, amplitude parameter in the transfer function, mutation radius on the weights, inverse temperature parameter in the softmax selection probability, mutation radius on the biases, reproduction ratio, mutation radius on the gain parameter of the transfer function, elitist ratio, number of feedforward pass to obtain the output, crossover ratio, upper loss threshold for early stop to trigger, reconstruction parameter, upper limit on the fraction of individual to have low loss to trigger early stop, upper limit on the best loss to trigger the addition of the cost term to the loss, population size, maximum number of generations before the execution is halted.

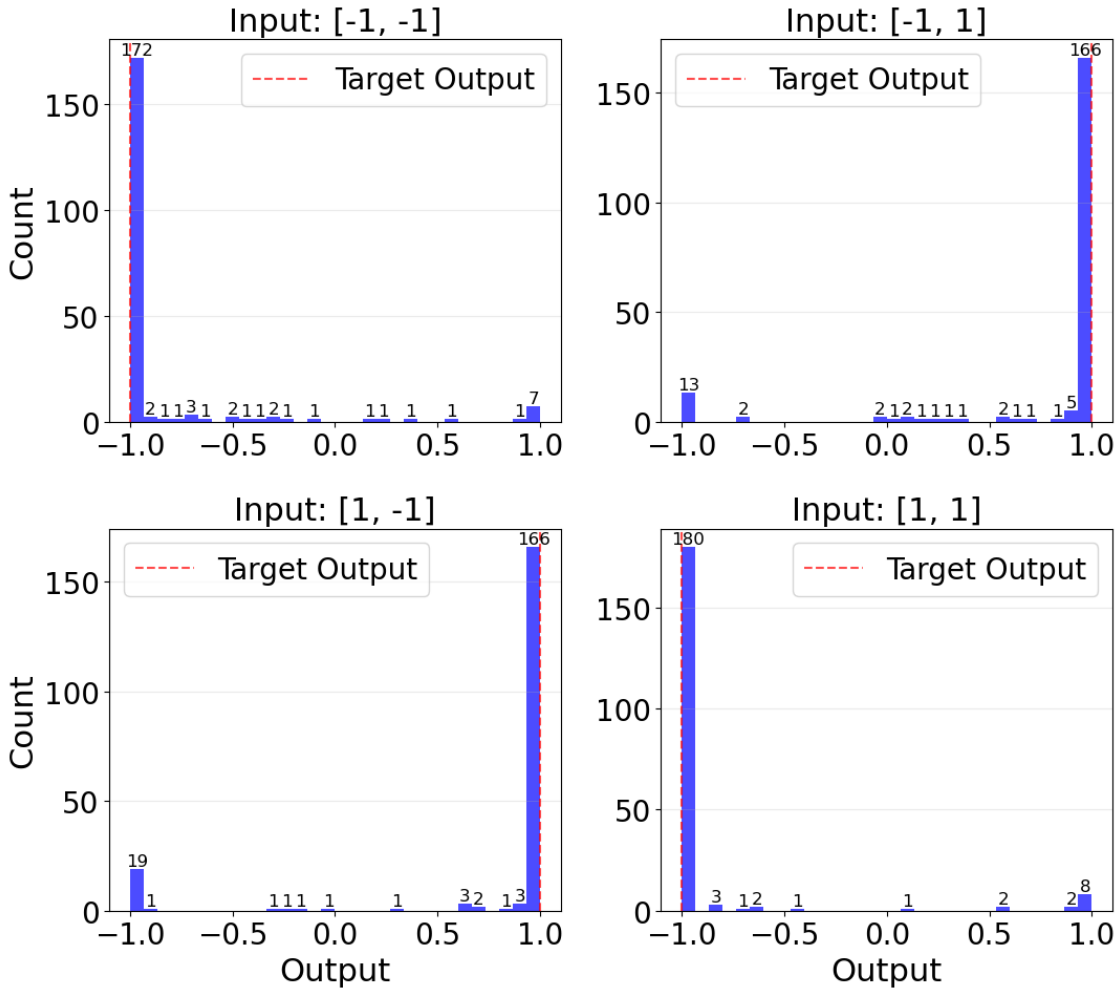

Figure 4.1: Outputs distributions on the final population for the 4 inputs. The vertical dashed red line indicates the target output given the input. The association input-target output is also reported in Table 4.1.

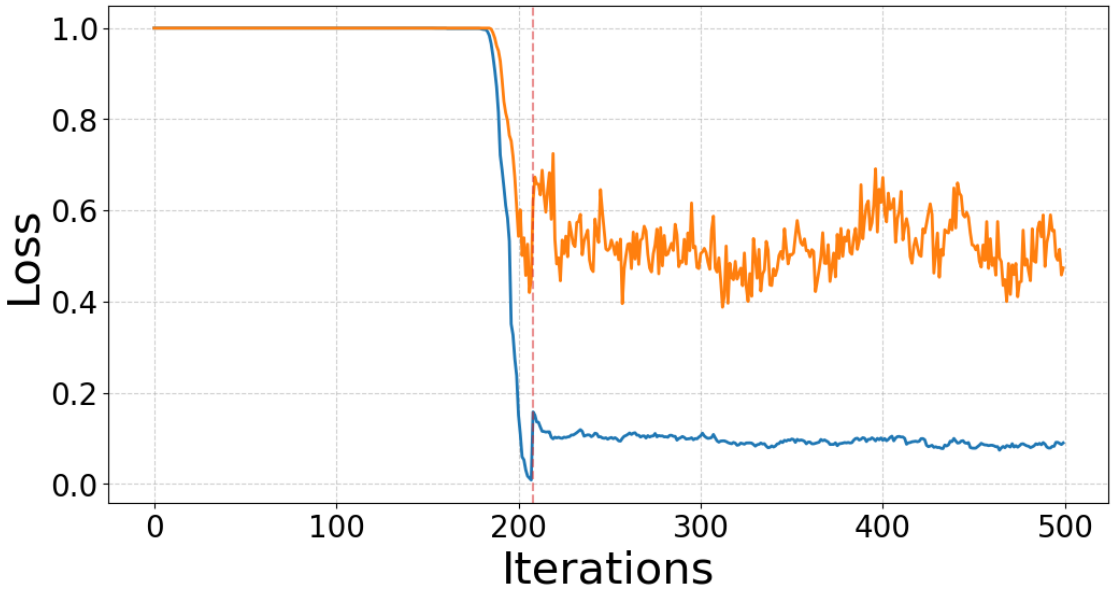

Figure 4.2: Loss evolution. In orange the mean loss over the population and in blue the best loss. The vertical red dashed line indicates when the cost term of the loss function was added.

Another interesting parameter to study is the connectivity and in particular the volume of links in the network and the 'expense' associated with the connectivity, represented by the cost term in the loss function (Subsection 4.1.3). Figure 4.3 (up) shows how, as asked by the cost in the loss function, the network population works towards minimizing their link density (or volume). This is demanded in the loss function as a proxy for the energetic cost of connectivity and the sensible tendency of evolution to favor efficiency. However, it can be noted how the mean volume starts decreasing early, largely before the introduction of the cost term in the loss function (indicated in Figure 4.3 by the vertical dashed red line). This result is not trivial since one would guess that a larger connectivity equals more processing power. On the contrary, this result shows that it is in the interest of the network to have a sparser connectivity seeing that too many links raises the difficulty of reaching a steady state. Then, when the cost term is introduced, the volume drops and stabilizes. It is significant to note that, even at lower volumes, the cost term is still driving the networks to lower their link density even further, but the stability reached proves that a lower volume would likely cause the network to fail the task.

On the other hand, the evolution of the mean cost over the population is intriguing since it suggests the opposite path forward for the networks in terms of connectivity. In fact, before the insertion of the cost term, while, although with different magnitudes,

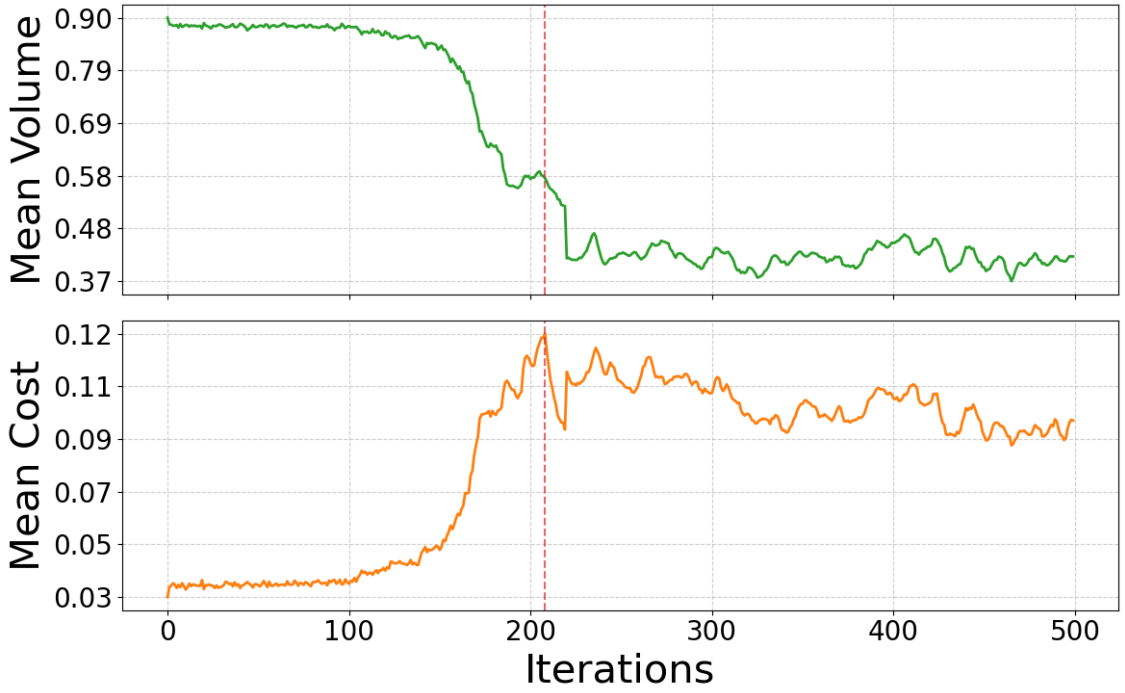

Figure 4.3: Mean volume and mean cost over the population. Top: mean volume of links over the population. Bottom: mean value of the cost term over the population. The vertical red dashed line indicates when the cost term of the loss function was added.

the mean volume decreases, the mean cost increases. However, the comparison of these results does not suggest a contradiction, but rather that, if left to only tackle the task, the networks would decrease their link density in favor of longer and stronger (i.e. with higher weights) connections. Moreover, after the addition of the cost term to the loss function, the cost stabilizes almost immediately. This seems to suggest that one of the possible reasons for the time the population took to solve the task is that is needed to evolve enough non-null connections in order to process the inputs.

What can be clearly stated from inspecting Figures 4.2 and 4.4 is that before $\sim$ 180 iterations, the cells with g significantly different from zero were still the minority (Figure 4.4); therefore, the output, given any input, was always zero; hence, the loss was not improving (Figure 4.2). Then, once the cells' transfer functions have developed the ability to produce an output (i.e. having the gain parameter larger than zero), the GA can actually start searching the parameter space.
It may not be obvious at first why the networks in the population take a considerably long time before producing any results. The explanation (other than the connectivity threshold mentioned in the previous paragraph) lies in both the selection process and

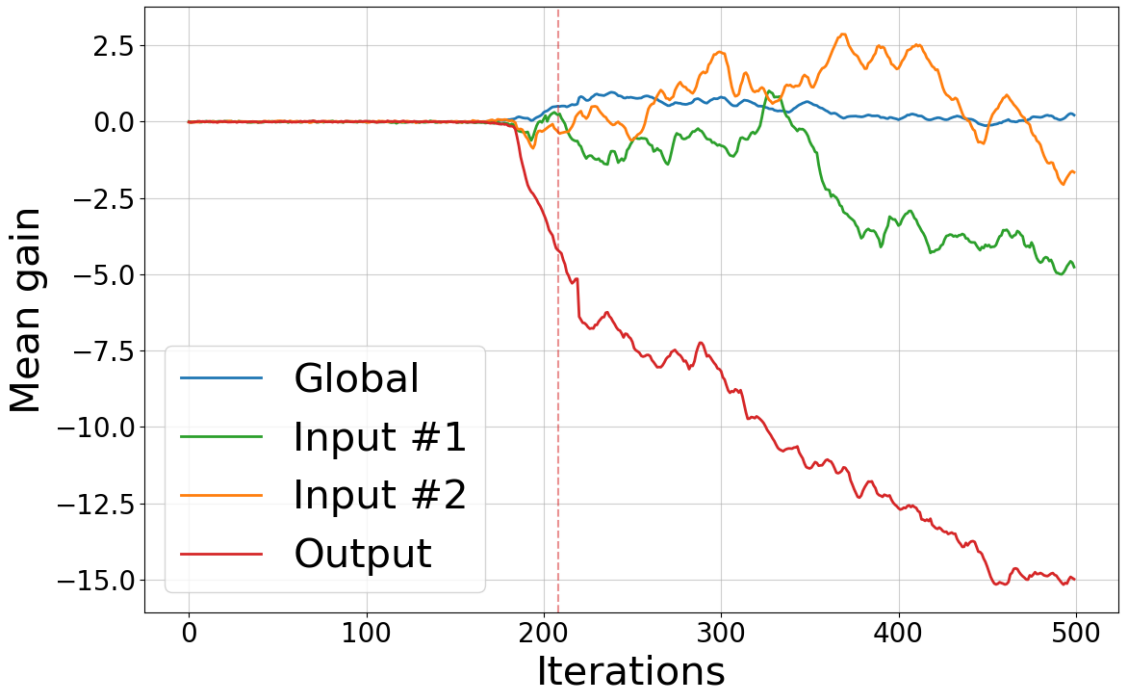

Figure 4.4: Evolution of the mean gain parameter over the population for various groups of cells for $\mu_T = 0.05$. The global mean gain parameter (blue curve), is the mean over the population of the mean gain parameter over all the cells of the network, with the exception of the output cell. The vertical red dashed line indicates when the cost term of the loss function was added.

the initialization technique: in fact, the networks are initialized with the gain parameter of each of their cells set to zero. Thus, an initialized network will not produce any output since any signal received by any of its cells will be flattened to zero due to the application of a transfer function that always returns zero [3]. Furthermore, the new individuals created through random generation are also initialized and will thus produce no output. Therefore, the only way for the population of networks to move out of the plateau of null gain parameter is to slowly evolve functional transfer functions through mutations of the gain parameter, which are inherited by reproduction (either with or without crossover recombination). Then, once some individuals are able to produce an output, the performance-based selection process will favor the reproduction of those individuals, quickly replacing the non-functional ones with their offspring.
This interpretation is verified by comparing Figures 4.4 and 4.5: indeed, in the first one, where the mutation radius for the gain parameter $\mu_T = 0.05$, the population evolves a functional transfer function around ~ 180 iterations, while in the second plot ($\mu_T = 0.1$),

[3]For clarification, see the definition of the transfer function Definition 4.1.4 and notice how, for $g \sim 0$, the curve is flat on the input axis.

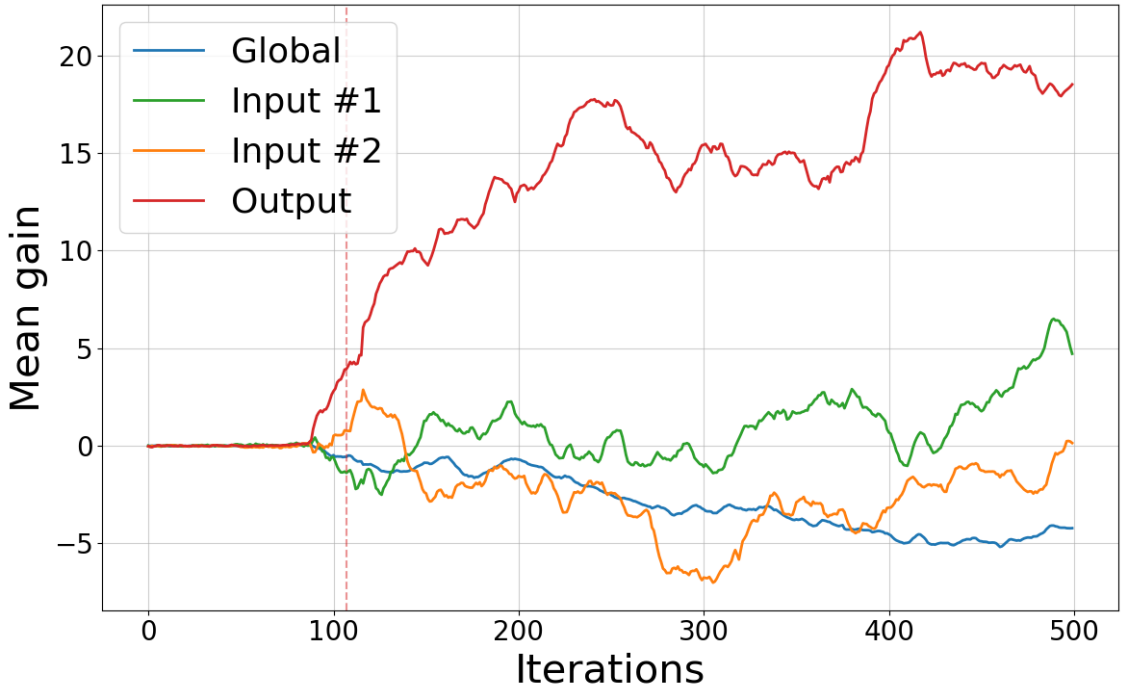

Figure 4.5: Evolution of the mean gain parameter over the population for various groups of cells for $\mu_T = 0.1$. The global mean gain parameter (blue curve), is the mean over the population of the mean gain parameter over all the cells of the network, with the exception of the output cell. The vertical red dashed line indicates when the cost term of the loss function was added.

it takes less than ~ 90 iterations [4]. However, other than in terms of velocity, employing a higher gain parameter's mutation radius μ_T does not improve performance.
Anyway, this behaviour is interesting also on an abstract level: indeed, it mimics what happens in the early stages of the evolution of life in a new environment (such as primordial Earth). In particular, an apparent possibility is the parallelism with the Cambrian explosion, i.e. the radiation of complex life that occurred in the early Paleozoic. In fact, before the Cambrian diversification, most organisms on Earth were relatively simple, and then, in the span of less than 25 million years, a surge in biological complexity occurred, generating most of the ancestors of the animals now present on the planet. Similarly, until the development of functional transfer functions, the networks that populate the simulation are fairly simple, with almost no capability of processing information. Then, quite suddenly, the population diversifies greatly, and the networks grow the ability to solve the rather complex given task.

[4] These two results are obtained through simulations with the same set of parameters (reported in Table 4.2) except for μ_T.

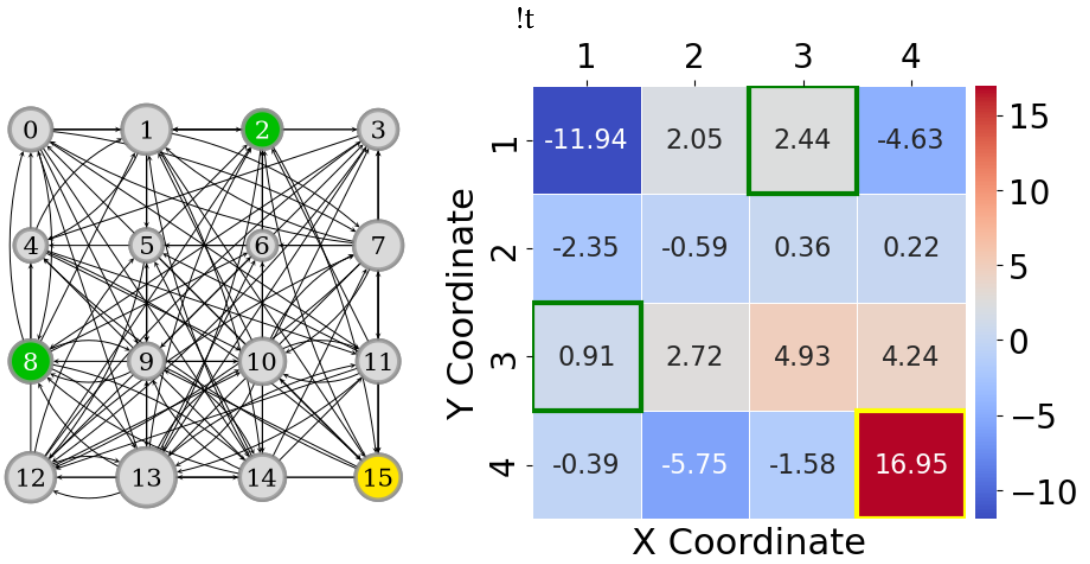

Figure 4.6: Graph structure and gain parameter distribution of the best solution in the population. The input cells are indicated in green (cell #2,8), while the output cell in yellow (cell #15). Left: graph of the network embedded in the squared lattice. The node size is proportional to its degree. Right: Heatmap of the gain parameter values on each cell of the network.

Furthermore, by looking at Figure 4.2, it can be noted how the evolution of the population follows directly from that of its best individual. In fact, it is clear how, at least qualitatively, the two plots are similar, presenting drops and rises at similar time periods. This is no surprise, since it was already apparent from the toy model how the search of a GA in the parameter space is led by the best solutions. Additionally, the same graph shows how the evolution of the population is not homogeneous, but rather erratic and characterized by large drops in loss, interspersed with plateaux. This behaviour is to be expected since the selection process employed in this model is loss-based and specifically favors the best-performing individuals. Besides, in this model, performing a parents selection '*with replacement*' finally matters. In fact, if, as often happens and is suggested by the results discussed in this paragraph, there is a group of individuals who are performing significantly better than the population's average, then the likelihood of the parents' population being formed mostly by repeating solutions from such an over-performing population's section is high.

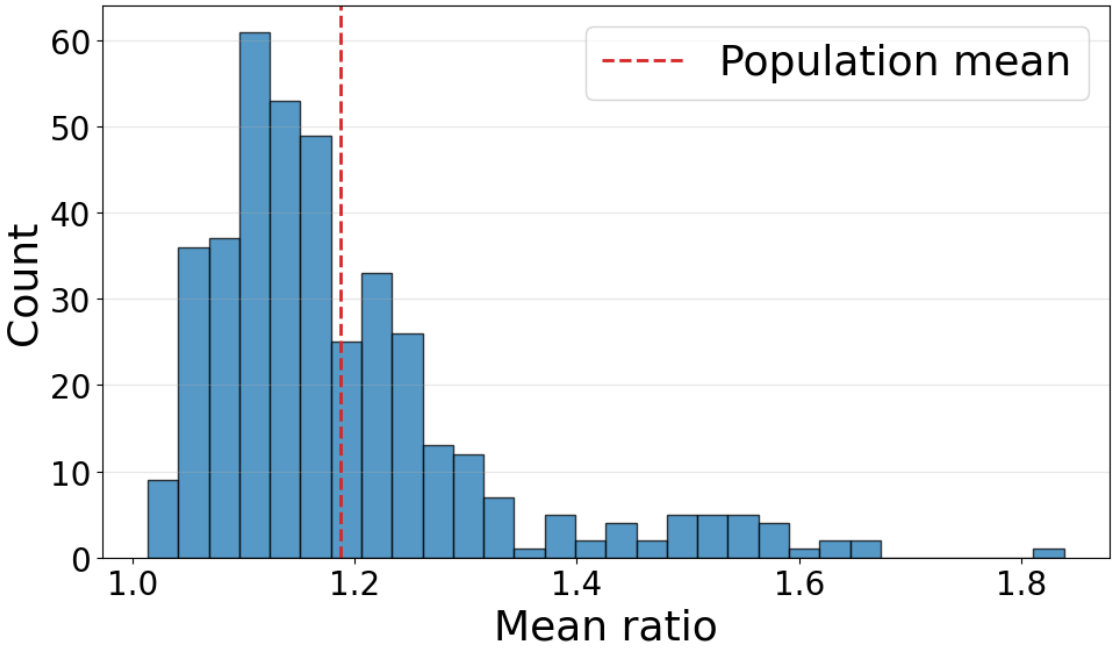

Figure 4.7: Distribution over the final population of the mean ratio between in-degree and out-degree of the cells in the network. The red dashed vertical line indicates the population mean. This result has been computed on a larger population size, namely $M = 400$.

5

Conclusions

This thesis studied the application of genetic algorithms to simulate natural evolution, focusing on the convergent evolution of neuron architectures.
In Chapter 3, the evolution of an artificial neural network was modeled through a GA, succeeding in the task of reproducing the XOR logic gate. In Chapter 4, a GA was implemented to simulate the evolution of excitable cells in a network.

The results of the analysis carried out in Chapter 3 show that the GA developed is capable of evolving an artificial neural network, replacing the usual training methods and opening possibilities for studying the evolution of parameters of an artificial neural network other than the weights and biases, such as the threshold of the non-linear activation function. The study presented in Chapter 3 laid the foundation for the main model discussed in the subsequent chapter.

The core of this thesis' work is the model presented in Chapter 4, where the basic GA developed in the previous chapter is expanded both in complexity and functionalities to construct a model dedicated to answering the two points of the main question presented at the start of this thesis in Chapter 1: is the structure of the neuron, as in an excitable cell with polarity in the information flow and a threshold response dynamics, a universal design principle?
The results presented in Chapter 4 have partially answered the two main questions of this study: from the distribution of the gain parameter, both in the best solution and in the whole population (mean quantities), it is evident that the transfer function does not converge to a step function (no *all-or-none* principle), with the exception of the output cell, which is expected to develop a threshold response dynamic, since the task requires a binary response. In most of the other cells in the network of the best solution in Figure 4.4 (right), the gain parameter g is relatively low and thus does not create threshold

response dynamics, but rather a continuous response.
On the other hand, regarding the emergence of polarity in the network, the graph in Figure 4.4 (left) suggests that no preferred direction has arisen in the connectivity Furthermore, Figure 4.7 shows that the cells in the networks of the final population have, on average, close to the same number of receiving and outgoing links, or, in other words, the ratio between the in-degree, which is the number of incoming links to the cell, and the out-degree, which is the number of outgoing links from the cell, is around 1. This indicates that no polarity, i.e. no preferred direction, has emerged from the evolution.

To conclude, this study found no significant evidence of the emergence of either polarity or threshold response dynamics, paving the way for further development in the field. On the other hand, the presented model successfully implements the evolution of excitable cells in a network through the completion of a given task and proposes a starting point for further advancement towards more precise modeling.

A

Fourier transfer function decomposition

In the model presented in Chapter 4, a fixed-shape transfer function (Definition 4.1.4) is employed, where the degree of freedom for evolution to optimize is the gain parameter, g, which regulates the slope of the function. This form of transfer function has the advantage of a simple implementation, while its development through the GA is still able to answer about whether a threshold response dynamics would emerge [1]. However, a more advanced modeling of the transfer function is possible.

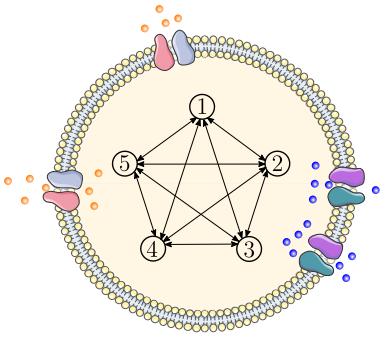

Figure A.1: Visual diagram of the internal dynamics of an excitable cell. An excitable cell transforms a biochemical input to an output signals. The cell response is determined by its internal dynamics. Image courtesy of Jordi Piñero.

Consider the macroscopic state of the excitable cell as the result of $n \in \mathbb{N}^+$ internal microscopic states (Figure A.1). It can be mapped to a scalar function, the cell's transfer function, which is decomposed into discrete Fourier modes:

$$\Phi_l(x) = \sum_{\nu=1}^{n} c_\nu^{(l)} e^{-2\pi i \nu x} , \qquad \text{(A.1)}$$

where $n \in \mathbb{N}$ is the number of internal states in cell l, while $\mathbf{c}$ is a vector of internal parameters of the cell that may evolve. This number serves as a proxy for the complexity in the response mech-

[1] The activation function implemented in the perceptron toy model in Chapter 3 does not offer this possibility since it implements a threshold response dynamics by definition.

anism of the cell: more internal states coincide with a larger processing and expression possibility. However, there is a cost associated with this complexity:

$$C_{resp} = \frac{1}{N} \sum_{k}^{N} \mathbf{c}^{(k)\top} \mathbf{c}^{(k)} \ . \tag{A.2}$$

This cost term would then be added to the loss function from Definition 4.1.3. In fact, it is a reasonable assumption that more complex response dynamics would be associated with a higher energy cost, thus justifying the request for its minimization through evolution.

The implementation of this transfer function form not only adds a cost term to the loss function, but also dramatically modifies the mutation mechanism, by acting on the components of $\mathbf{c}$. In particular, the process unfolds as follows:

1. Discretize the interval $[A, B]$, thus obtaining a set $\{x_k\}$ of n of evenly spaced elements in it.
2. Apply Φ on $\{x_k\} \Rightarrow \Phi(x_k) = \{\Phi_k\}$.
3. Compute Direct Fourier Transform (DFT) on $\{\Phi_k\}$, obtaining the complex Fourier coefficients:
$$c_k = \sum_{v=0}^{n-1} \Phi_v \, e^{-i2\pi \frac{kv}{n}}, \quad \forall k \in [0, n), \quad c_k \in \mathbb{C}$$
which are the components of vector $\mathbf{c} = \{c_k\}$.
4. Mutate/evolve every elements of $\{c_k\}$ as for the gain parameter g, thus obtaining a new set of (mutated) coefficients, $\{c'_k\}$.
5. Inverse DFT on $\{c'_k\}$:
$$\Phi'_k = \frac{1}{n} \sum_{v=0}^{n-1} c'_k \, e^{i2\pi \frac{kv}{n}}$$
6. Fit $\{\Phi'_n\}$ with a polynomial of grade s to obtain the new transfer function Φ'.

This algorithm overwrites the mutation procedure on the transfer function in the mutation step of the GA.
Note that, in this version, the transfer function Φ_l of neuron π_l is initialized as a null function $\Phi_l(x) = 0$, as for the fixed-shape transfer function employed in the model.

Thus, the n components $\{c_k^{(l)}\}$ will initially be all null and then will become non-null through mutations.

B

Code development

The results presented in this thesis are produced through Python simulations. All the code resources are available in a public GitHub repository at [70]. Some simulations have been carried out on an Ubuntu virtual machine provided by Cloudveneto (https://cloudveneto.it/) through the resources of the University of Padova.

Bibliography

[1] George R. McGhee. *Convergent Evolution: Limited Forms Most Beautiful*. Cambridge, MA: MIT Press, 2011.

[2] David E Alexander. *On the wing: insects, pterosaurs, birds, bats and the evolution of animal flight*. Oxford University Press, 2015.

[3] David L. Williams. "Evolution of Photoreception and the Eye". In: *Wild and Exotic Animal Ophthalmology: Volume 1: Invertebrates, Fishes, Amphibians, Reptiles, and Birds*. Ed. by Fabiano Montiani-Ferreira, Bret A. Moore, and Gil Ben-Shlomo. Cham: Springer International Publishing, 2022, pp. 3–7. DOI: 10.1007/978-3-030-71302-7_1.

[4] Ben P Williams et al. "Phenotypic landscape inference reveals multiple evolutionary paths to C_4 photosynthesis". In: *eLife* 2 (Sept. 2013). Ed. by Dominique Bergmann, e00961. DOI: 10.7554/eLife.00961.

[5] Ricard Solé et al. "Fundamental constraints to the logic of living systems". In: *Interface Focus* 14.5 (Oct. 2024), p. 20240010. ISSN: 2042-8898. DOI: 10.1098/rsfs.2024.0010.

[6] Leonid L. Moroz and Peter A. V. Anderson. "Convergent evolution of neural systems in ctenophores". In: *Journal of Experimental Biology* 218.4 (Feb. 2015), pp. 598–611. ISSN: 0022-0949. DOI: 10.1242/jeb.110692.

[7] Pedro Martinez and Simon G. Sprecher. "Of Circuits and Brains: The Origin and Diversification of Neural Architectures". In: *Frontiers in Ecology and Evolution* 8 (2020). ISSN: 2296-701X. DOI: 10.3389/fevo.2020.00082.

[8] Stephen Jay Gould. *Wonderful Life: The Burgess Shale and the Nature of History*. New York, NY: W. W. Norton & Company, 1989.

[9] David M Raup. "Geometric analysis of shell coiling: general problems". In: *Journal of paleontology* (1966), pp. 1178–1190.

[10] Conrad H Waddington. "Canalization of development and the inheritance of acquired characters". In: *Nature* 150.3811 (1942), pp. 563–565. DOI: https://doi.org/10.1038/150563a0.

[11] Stuart A. Kauffman and Andrea Roli. "A third transition in science?" In: *Interface Focus* 13.3 (Apr. 2023), p. 20220063. ISSN: 2042-8898. DOI: 10.1098/rsfs.2022.0063.

[12] Michael Begon and Colin R Townsend. *Ecology: from individuals to ecosystems*. John Wiley & Sons, 2020.

[13] University of California, Berkeley. *Homologies and Analogies*. https://evolution.berkeley.edu/evo101/IIE2bHomologies.shtml. Archived from the original on 19 November 2016. 2016. (Visited on 01/10/2017).

[14] University of California, Berkeley. *Analogy: Squirrels and Sugar Gliders*. https://evolution.berkeley.edu/evo101/IIE2b2Analogy.shtml. Archived from the original on 27 January 2017. 2017. (Visited on 01/10/2017).

[15] Erik S Wright. "Tandem repeats provide evidence for convergent evolution to similar protein structures". In: *Genome Biology and Evolution* 17.2 (2025), evaf013. DOI: 10.1093/gbe/evaf013.

[16] Leonid L Moroz et al. "The ctenophore genome and the evolutionary origins of neural systems". In: *Nature* 510.7503 (2014), pp. 109–114. DOI: 10.1038/nature13400.

[17] Joseph F Ryan et al. "The genome of the ctenophore Mnemiopsis leidyi and its implications for cell type evolution". In: *Science* 342.6164 (2013), p. 1242592.

[18] William B Kristan. "Early evolution of neurons". In: *Current Biology* 26.20 (2016), R949–R954.

[19] Dale Purves et al. "Voltage-gated ion channels". In: *Neuroscience. Sinauer Associates, Sunderland* (2001).

[20] Mark W Barnett and Philip M Larkman. "The action potential". In: *Practical neurology* 7.3 (2007), pp. 192–197.

[21] Detlev Arendt. "Elementary nervous systems". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 376.1821 (2021).

[22] Michael G. Paulin and Joseph Cahill-Lane. "Events in Early Nervous System Evolution". In: *Topics in Cognitive Science* 13.1 (2021), pp. 25–44. DOI: https://doi.org/10.1111/tops.12461.

[23] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[24] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[25] Leon Glass and Stuart A Kauffman. "The logical analysis of continuous, non-linear biochemical control networks". In: *Journal of theoretical Biology* 39.1 (1973), pp. 103–129.

[26] KE Kurten. "Correspondence between neural threshold networks and Kauffman Boolean cellular automata". In: *Journal of Physics A: Mathematical and General* 21.11 (1988), p. L615.

[27] Bartolo Luque and Ricard V Solé. "Phase transitions in random networks: Simple analytic determination of critical points". In: *Physical Review E* 55.1 (1997), p. 257.

[28] Stefan Bornholdt. "Boolean network models of cellular regulation: prospects and limitations". In: *Journal of the Royal Society Interface* 5.suppl_1 (2008), S85–S94.

[29] Giorgio Parisi. "A simple model for the immune network." In: *Proceedings of the National Academy of Sciences* 87.1 (1990), pp. 429–433.

[30] Alan S Perelson and Gérard Weisbuch. "Immunology for physicists". In: *Reviews of modern physics* 69.4 (1997), p. 1219.

[31] Elena Agliari et al. "Anergy in self-directed B lymphocytes: a statistical mechanics perspective". In: *Journal of theoretical biology* 375 (2015), pp. 21–31.

[32] Jean-Louis Deneubourg et al. "The dynamics of collective sorting robot-like ants and ant-like robots". In: *From animals to animats: proceedings of the first international conference on simulation of adaptive behavior*. 1991, pp. 356–365.

[33] Ricard V Solé, Octavio Miramontes, and Brian C Goodwin. "Oscillations and chaos in ant societies". In: *Journal of theoretical Biology* 161.3 (1993), pp. 343–357.

[34] Guy Theraulaz et al. "Spatial patterns in ant colonies". In: *Proceedings of the National Academy of Sciences* 99.15 (2002), pp. 9645–9649.

[35] Iain D Couzin. "Collective cognition in animal groups". In: *Trends in cognitive sciences* 13.1 (2009), pp. 36–43.

[36] Patrick McMillen and Michael Levin. "Collective intelligence: A unifying concept for integrating biology across scales and substrates". In: *Communications Biology* 7.1 (2024), p. 378.

[37] Tahir I Yusufaly and James Q Boedicker. "Mapping quorum sensing onto neural networks to understand collective decision making in heterogeneous microbial communities". In: *Physical biology* 14.4 (2017), p. 046002.

[38] Ricard Solé, Melanie Moses, and Stephanie Forrest. *Liquid brains, solid brains*. 2019. DOI: 10.1098/rstb.2019.0040.

[39] Jordi Piñero and Ricard Solé. "Statistical physics of liquid brains". In: *Philosophical Transactions of the Royal Society B* 374.1774 (2019), p. 20180376.

[40] Uri Alon. *An introduction to systems biology: design principles of biological circuits*. CRC press, 2019.

[41] Kim Sneppen. *Models of life*. Cambridge University Press, 2014.

[42] Simona Ginsburg and Eva Jablonka. "The evolution of associative learning: A factor in the Cambrian explosion". In: *Journal of theoretical biology* 266.1 (2010), pp. 11–20.

[43] Gualtiero Piccinini. "The First computational theory of mind and brain: a close look at mcculloch and pitts's "logical calculus of ideas immanent in nervous activity"". In: *Synthese* 141 (2004), pp. 175–215.

[44] Shuichi Shigeno. "Brain evolution as an information flow designer: The ground architecture for biological and artificial general intelligence". In: *Brain Evolution by Design: From Neural Origin to Cognitive Architecture* (2017), pp. 415–438.

[45] Shuichi Shigeno et al. "Cephalopod brains: an overview of current knowledge to facilitate comparison with vertebrates". In: *Frontiers in Physiology* (2018), p. 952.

[46] Pablo Garcia-Lopez, Virginia Garcia-Marin, and Miguel Freire. "The histological slides and drawings of Cajal". In: *Frontiers in neuroanatomy* 4 (2010), p. 1156.

[47] Geoffrey North and Ralph J Greenspan. *Invertebrate neurobiology*. Cold Spring Harbor Laboratory Press, 2007.

[48] Daniel C Dennett. *Kinds of minds: Toward an understanding of consciousness*. Basic Books, 2008.

[49] Russell Powell et al. "Convergent minds: the evolution of cognitive complexity in nature". In: *Interface Focus* 7.3 (Apr. 2017), p. 20170029. ISSN: 2042-8898. DOI: 10.1098/rsfs.2017.0029.

[50] Daniel C Dennett. *From bacteria to Bach and back: The evolution of minds*. WW Norton & Company, 2017.

[51] Ricard Solé and Luís F Seoane. "Evolution of brains and computers: the roads not taken". In: *Entropy* 24.5 (2022), p. 665.

[52] Aina Ollé-Vila, Luís F. Seoane, and Ricard Solé. "Ageing, computation and the evolution of neural regeneration processes". In: *Journal of The Royal Society Interface* 17.168 (July 2020), p. 20200181. ISSN: 1742-5689. DOI: 10.1098/rsif.2020.0181.

[53] Stephen Whitelam et al. "Correspondence between neuroevolution and gradient descent". In: *Nature communications* 12.1 (2021), p. 6317. DOI: 10.1038/s41467-021-26568-2.

[54] Gregory Hornby et al. "Automated antenna design with evolutionary algorithms". In: *Space 2006*. 2006, p. 7242.

[55] Claus Aranha and Hitoshi Iba. "Application of a Memetic Algorithm to the Portfolio Optimization Problem". In: *AI 2008: Advances in Artificial Intelligence*. Ed. by Wayne Wobcke and Mengjie Zhang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 512–521.

[56] David G Mayer. *Evolutionary algorithms and agricultural systems*. Springer, 2002.

[57] Junchen Jin, Xiaoliang Ma, and Iisakki Kosonen. "A stochastic optimization framework for road traffic controls based on evolutionary algorithms and traffic simulation". In: *Advances in Engineering Software* 114 (2017), pp. 348–360. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2017.08.005.

[58] Shixiang Sun et al. "Inducible aging in Hydra oligactis implicates sexual reproduction, loss of stem cells, and genome maintenance as major pathways". In: *GeroScience* 42.4 (2020), pp. 1119–1132. DOI: https://doi.org/10.1007/s11357-020-00214-z.

[59] Marvin Minsky and Seymour Papert. "An introduction to computational geometry". In: *Cambridge tiass., HIT* 479.480 (1969), p. 104.

[60] Seppo Linnainmaa. "Taylor expansion of the accumulated rounding error". In: *BIT Numerical Mathematics* 16.2 (1976), pp. 146–160. DOI: https://doi.org/10.1007/BF01931367.

[61] James W Kalat and Elaine M Hull. *Biological psychology*. Thomson/Wadsworth Belmont, CA, 2007.

[62] E Kramarsky-Winter and Y Loya. "Regeneration versus budding in fungiid corals: a trade-off". In: *Marine Ecology Progress Series* 134 (1996), pp. 179–185. DOI: https://doi.org/10.3354/meps134179.

[63] Scott Fahlman and Christian Lebiere. "The cascade-correlation learning architecture". In: *Advances in neural information processing systems* 2 (1989).

[64] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[65] Frank Rosenblatt et al. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Vol. 55. Spartan books Washington, DC, 1962.

[66] Rahul Sarpeshkar. "Analog Versus Digital: Extrapolating from Electronics to Neurobiology". In: *Neural Computation* 10.7 (Oct. 1998), pp. 1601–1638. ISSN: 0899-7667. DOI: 10.1162/089976698300017052.

[67] Gáspár Jékely. "The chemical brain hypothesis for the origin of nervous systems". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 376.1821 (Feb. 2021), p. 20190761. ISSN: 0962-8436. DOI: 10.1098/rstb.2019.0761.

[68] Shun Ogawa, Francesco Fumarola, and Luca Mazzucato. "Multitasking via baseline control in recurrent neural networks". In: *Proceedings of the National Academy of Sciences* 120.33 (2023), e2304394120. DOI: 10.1073/pnas.2304394120.

[69] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Feb. 1996. ISBN: 9780262280013. DOI: 10.7551/mitpress/3927.001.0001.

[70] Pietro Malagoli. *Convergent evolution of neuron architectures*. 2026. DOI: 10.5281/zenodo.19340521.