Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Pavel John

Spatial modeling of brain tissue

Department of Theoretical Computer Science
and Mathematical Logic

Supervisor of the master thesis: Mgr. Roman Neruda, CSc.

Study programme: Computer Science

Specialization: Nonprocedural programming and artificial intelligence

Prague 2013

I dedicate this thesis to my family for their endless love and support. The knowledge, that there is always a place I can return to, gives me the confidence to take up much grater challenges than I could ever do only by myself.

I would like to thank my supervisor Mgr. Roman Neruda, CSc. for his guidance and discussions.

Název práce:
Spatial modeling of brain tissue

Autor:
Pavel John

Katedra:
Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce:
Mgr. Roman Neruda, CSc., Ústav informatiky AV ČR, v.v.i.

Abstrakt:
Nervová spojení v lidském mozku se mění na základě vjemů z okolí. Způsob, jakým k proměnám dochází, a jak přesně tyto proměny ovlivňují vlastnosti mozkové tkáně, dosud není zcela pochopen. Práce zkoumá souvislost paměti a učení s prostorovým uspořádáním neuronů, zejména pak s tvarem jejich dendritických výběžků. Součástí je model, který zachycuje mozkovou tkáň pomocí dvourozměrné mřížky s různými druhy spojení mezi jednotlivými buňkami mřížky. Tento model je formálně definován a dále podroben teoretickému zkoumání. Zásadním výsledkem je důkaz věty o výpočetní síle definovaného modelu na úrovni Turingova stroje. K nalezení vhodné architektury vzhledem k problému slouží několik variant evolučních algoritmů. Model s danou architekturou je dále adaptován na základě komunikace s prostředím. Popsané myšlenky jsou implementovány a podrobeny několika experimentům, které poukazují na důležité vlastnosti modelu.

Klíčová slova:
Neuronové sítě, Celulární automaty, Kognitivní vědy, Evoluční algoritmy

Title:
Spatial modeling of brain tissue

Author:
Pavel John

Department:
Department of Theoretical Computer Science and Mathematical Logic

Supervisor:
Mgr. Roman Neruda, CSc., Institute of Computer Science of the ASCR, v.v.i.

Abstract:
Neural connections in the human brain are known to be modified by experiences. Yet, little is known about the mechanism of the modification and its implications on the brain function. The aim of this thesis is to investigate what impact the spatial properties of brain tissue can have on learning and memory. In particular, we focus on the dendritic plasticity. We present a model where the tissue is represented by a two-dimensional grid and its structure is characterized by various connections between the grid cells. We provide a formal definition of the model and we prove it to be computational as strong as the Turing machine. An adaptation algorithm proposed enables the model to reflect the environmental feedback, while evolutionary algorithms are employed to search for a satisfactory architecture of the model. Implementation is provided and several experiments are driven to demonstrate the key properties of the model.

Keywords:
Neural networks, Cellular automata, Cognitive science, Evolutionary algorithms

Contents

Introduction

The idea of brain as a center of intellect is very old. It is known that Hippocrates (460-379 B.C.) stated his belief that the brain was the seat of intelligence [2]. It took more than two thousand years till the nineteenth century, when Camillo Golgi discovered a stain that colored some of the brain cells, called neurons, and revealed their structure.

A neuron is composed of a cell body, named soma, and many thin tubes that radiate from the soma, called neurites. There are two types of neurites, axons and dendrites. The former can be even a meter long and are believed to carry output from a neuron to its targets. The latter are usually shorter then two millimetres and they act like antennae of the neuron to receive the signals.

That picture led histologist Santiago Ramón y Cajal to the conclusion, that neurons don't form a continuous system like blood vessels as it was believed by that time. Instead he stated that the neurons must communicate by contact. Since then, the scientists are trying to figure out what signals can be measured on various places of the neural body and what is their exact meaning. To illustrate what was available by Cajal's time, see the Figure 1.

The paradigm of many communicating cells combined with further discoveries motivated Warren McCulloch and Walter Pitts to create a mathematical model of a neural network [14]. Theoretical modelling of brains pursue two goals since then. The first is to understand the mechanisms of the brain like thinking and memory, while the second takes the biological knowledge as an inspiration for statistical tools designed to solve complex problems of the real world.

The first target is pursued in this work and the second is used to evaluate the results. The aim is to study some of the issues of understanding the organic brain by means of computer science. For that reason, we propose a theoretic model that enables us to transform the problems from biological environment to the field of computer science. We then examine the theoretical capabilities of the model using tools of the automata theory. Next we propose methods for searching the right configuration of the model by means of evolutionary algorithms and we design a mechanism for adaptation of the model when solving a given problem. We provide several links between our theoretical study and the mechanisms that are observed in nature. Based on the methods proposed in this work and their comparison to biological knowledge, we offer a possible explanation of some of the learning processes that happen in the real brain.

To demonstrate properties of our model in practice, we provide a software implementation and we run several experiments. The model is put into a synthetic environment where it solves elementary logical tasks. This way we use the second branch of brain modelling, where the models are used perform specific computations, to verify that the model does what we expect it to do.

We focus on the spatial properties of the brain tissue in our study. It seems that one of the factors which form our memory is the shape of dendrites [2]. We recall that dendrites are branched projections of neurons and that they are supposed to act like antennae for receiving signal travelling between neurons. The shape of dendrites is very likely to be highly dynamic and to be formed with respect to experience [13], [28].

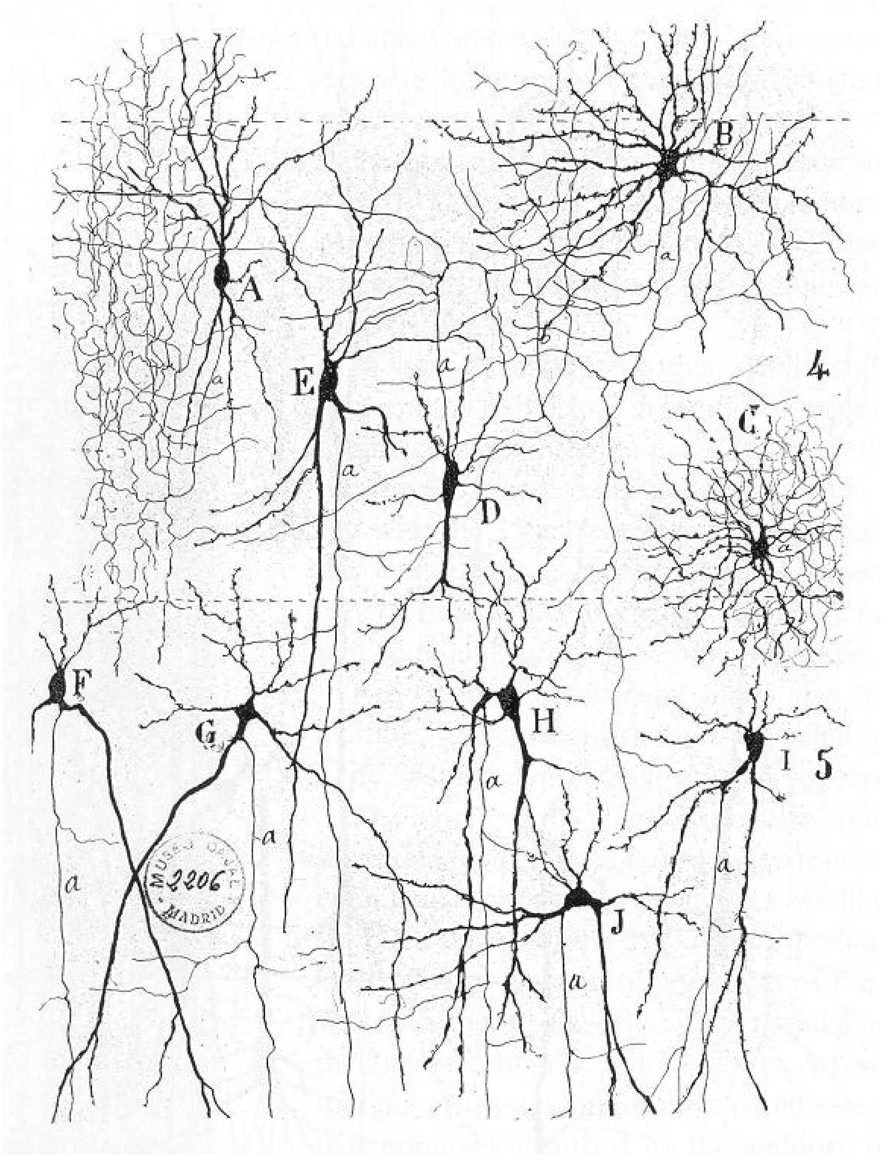


Figure 1: From “Texture of the Nervous System of Man and the Vertebrates” by Santiago Ramón y Cajal. The figure illustrates the diversity of neuronal morphologies in the auditory cortex.

Our model describes the dynamics with high level of abstraction and simplification. We conceive a two-dimensional space and we discretize it to form a grid. Cells of the grid are connected in various ways between each other and there are also connections to and from the outside of the grid. Different level of electric potential is considered to be present in distinct locations of the grid. Strength of some of the connections is modelled by conductance, that is, by the speed of equalization of the potential between neighboring locations. For that reason, we call our model the *conductive field brain*.

All the connections and the grid itself describe the modelled brain tissue, namely the dendrites, somata, and axons of neurons that form it. There are many other types of cells which compose the brain tissue, like the neuroglia for example [2], but we consider only the neurons for the sake of simplicity. The goal is to figure out what shall be the rules for connecting the different locations on the grid, and, preferably, to induce the laws followed by the process that shapes these connections.

We explore the state of the art in the chapter 1 and we compare our ideas to what is already available. The model is described in detail in the Chapter 2, which enables us to study some principles that the brain could follow. To support the choice of the model, we examine it from the theoretical point of view in the Section 2.2. The Chapter 3 explains the issue of searching the right configuration of the model. Then we implement the model as a piece of software and we describe it the Chapter 4. And finally, several experimental results are delivered in the Chapter 5.

1. Related work

In this section, we compare the conductive field brain to the most successful and deeply studied computational models that incorporate the principles used in the model. The most resembling are various types of neural networks and cellular automata. We also take into account several works that combine the notion of grid and connectionist models.

To begin with neural networks, we firstly need at least a vague definition of what is a neural network. To meet this request, we cite Raúl Rojas: "If we conceive of each node in an artificial neural network as a primitive function capable of transforming its input in a precisely defined output, then artificial neural networks are nothing but networks of primitive functions" [19]. He than adds that various models differ in the primitive functions used and the interconnection pattern.

Our model contains a grid of real numbers, which is interconnected on a neighborhood principle and the values are equalized in time. That is a high-dimensional dynamic system itself, not considering other connections present. If we were to say whether the conductive field brain is a neural network, we would incline to the answer no. The model satisfies the condition that the neurons are primitive functions with deterministic output, but we find the interconnection too complex to call it just a network of neurons. In contrast, it may take the signal emitted by an axon several steps till it reaches its target and the signal may combine with diverse concurrent signals by the way.

The conductive field brain doesn't seem to be a neural network from inside. We compare its properties to neural networks from outside. It takes a vector of inputs and produces a vector of outputs every step. Feed-forward networks don't carry any information between two consecutive computations. Therefore it is meaningful to compare the conductive field brain to recurrent networks. Both models keep an intrinsic state. As we show in the Section 2.2, the conductive field brain is at least as strong as a finite state machine or the Turing machine, depending on the context. The recurrent networks can also simulate a finite state machine [19] and as stated by Siegelmann [22], they can have super-Turing properties in a specific environment. Therefore, the conductive field doesn't bring any additional theoretical computing power. It can bring, however, some additional insight to a spatial dependant learning process.

Candidate model for a comparison is also the Hopfield network, where the neurons are represented by a set of binary threshold computing units, and every pair of the units is interconnected. One of the major learning methods for Hopfield networks is inspired by the Hebbian principle [19]. The principle is sometimes summarized by "what fire together, wire together". This shortcut could, in general, fit also to the adaptation methods presented in this work, although the details are different and the modification is modulated by multiple factors. More details on that topic are presented in the Chapter 3. However we can find some similarities, the Hopfield networks and the conductive field brain are conceptually different. They differ in the way the threshold units are interconnected and the notion of self-organisation is not present in the conductive field brain, since it needs feedback to adapt.

Because of the grid interconnected on the neighborhood principle, we can try to find some similarities in comparison to cellular automata. They also work in regular formation and they determine the next state of every cell on the basis of the current state of the cell and on the sates of its neighborhood. The topic is well-studied and is described by Alonso-Sanz [1].

Nevertheless, the cellular automata lack sparse irregular connections over long distances that are present in the conductive field brain. This issue is partially solved by considering cellular automata networks proposed by Yang and Yang [27]. They modify the conventional cellular automata by adding shortcut connections and then they study properties of the newly created structure. Yet, the shortcuts obey the same rules as the neighborhood connections while the conductive field brain uses the threshold units. The threshold units perform a lot of computation and they communicate with input and output. Since there is no such tool in the cellular automata networks, the models become incomparable in their function.

An other combination of the ideas of cellular automata and neural networks was presented by Chua and Yang [4], [5] and reviewed by Cimagalli and Balsi [6] five years later. It is a neural network interconnected in the same structure as a cellular automata. The model is called a cellular neural network. Yet, it is still a neural network with simple connections between neurons though in a regular structure.

To design our model to more detail, we also searched for solutions of related problems. If we conceive the potential values in the conductive field as column heights of some kind of liquid, we can find some analogy in liquid surface modelling. That topic is examined by Di Gregorio and Serra [7], who apply their theoretical models to debris or mud flow simulations. Parsons and Fonstad [16] model a surface of water by cellular automata. We do not implement to our model any of the methods exactly, but we take inspiration from the solutions proposed in these two papers.

To sum up this section we conclude that there is a wide variety of combinations of the two key ideas. The grid where local information is processed in every cell and than communicated with its neighborhood is mostly called cellular automata or sometimes dynamic systems in case of continuous states. It is mainly used to simulate complex systems or to process an information on a locality basis. And next, there are neural networks with an idea of primitive functions wired together to form a network with an input and an output. We wish to contribute to understanding of these structures and to come up with a notion of how the primitive functions may be connected on the locality basis. Our model uses the neural network idea and replaces the wired connections by an environment simulated by cellular automata. The model is designed to bring some insight into the biological knowledge of human brain eventually.

2. The model

For the purposes of the model, we accept the paradigm that electrical impulses carry the substance of the information coded in the brain. We also think of neurons as of simple computational units consisting of a soma, dendrites and an axon. The soma decides whether the neuron produces electrical signal or not. The dendrites carry the signal from the surrounding environment to the soma and the axon transports the signal from the soma to a different, sometimes distant, place.

Lets imagine a schema of a neural soma and the dendrites radiating from it as depicted in the Figure 2.1 a). We concentrate our attention to its spatial properties. For the purposes of the model, we simplify the situation by taking into account only two dimensions. Then we discretize the problem to obtain a grid. This enables us to see the environment as regularly distributed cells, that have defined different properties of conductance of signal between them. The cells that represent places connected by a thick dendrite are supposed to share the signal easily from one to an other. The cells that have a thin or none dendrite between them are assumed to conduct the signal weakly. Moreover, the discretization has its consequences in the representation of time. We think of the model as of a synchronized system working in steps. The idea of discretization is depicted in the Figure 2.1.

We now describe a model of a brain tissue containing somata and dendrites represented by a grid. The axons are realized by a cell more or less distant from the soma. The grid has a parameter for every cell, called a potential. If the potential in a cell that stands for a soma raises above a certain threshold, the soma decides to produce a signal. This results into a change of potential level in both the somatic and the axonal cell. The potential in the somatic cell is always diminished by a spike. Axon changes the potential either in a positive or a negative way, but every neuron keeps the same polarity forever. We call the neurons that produce positive signal excitatory and those producing negative signal are inhibitory.

The neurons we have described so far have both the soma and the axon inside the grid and are therefore named internal. There are also neurons that wire the model to the outer world by inputs called sensors and outputs called motors. Sensors are additional places, where neurons of a special kind, called sensory neurons, have their somata. The input consists of boolean values, one for each sensor. It says whether a group of sensory neurons connected to the sensor is active or not. The output works in a similar manner. A motor neuron has its soma inside a cell of the grid and the axon is connected to an output, called motor. An illustrative example is depicted in the Figure TODO

Because the model is inspired by mechanisms observed in human brain and the main difference from neural networks is the grid of cells that conducts the potential, we call the model a *conductive field brain.* The description provided in this chapter goes to the extent needed for classification of its general properties. We give formal definition and more detailed description in the following chapters.

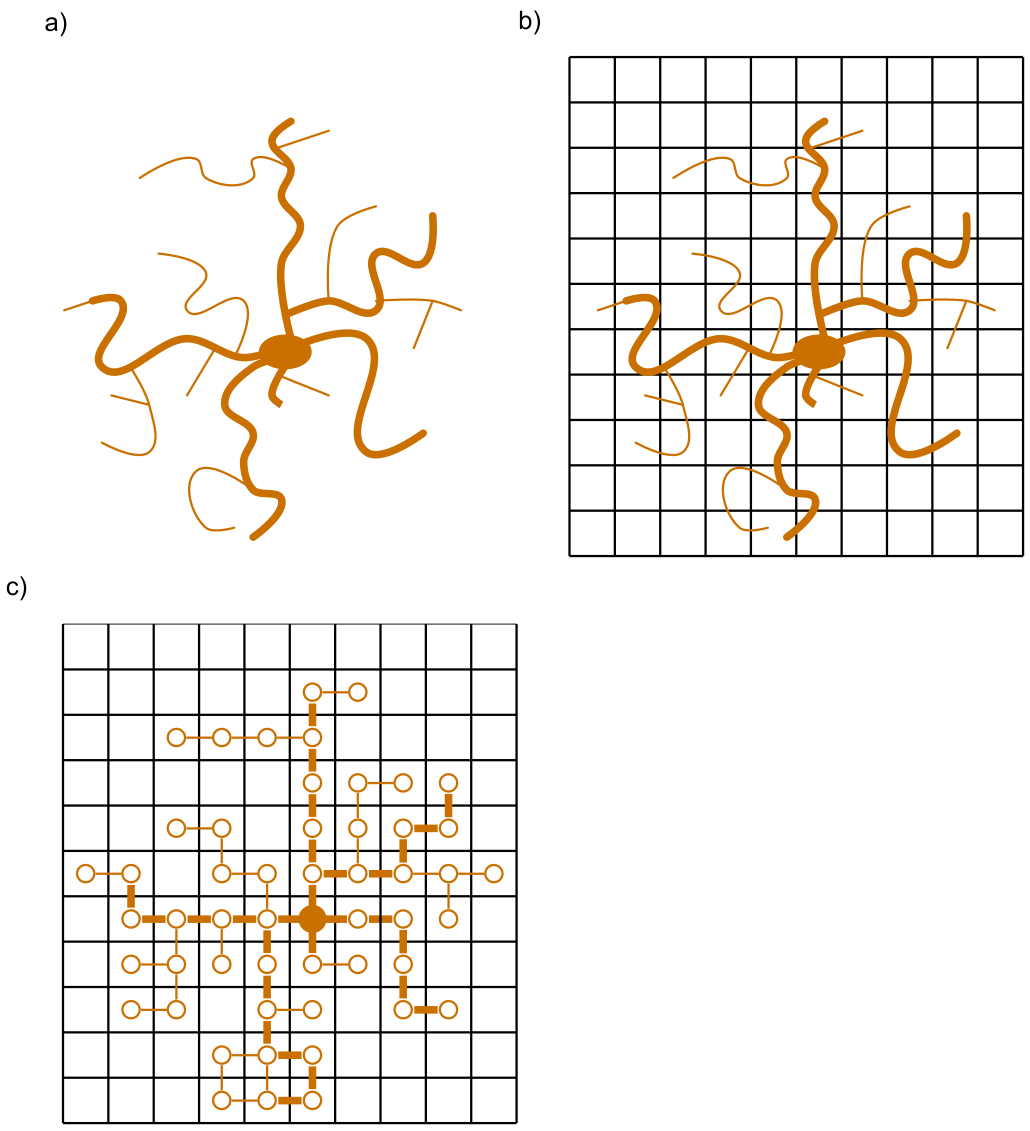


Figure 2.1: Schematic illustration of a) a neuron; b) a neuron on a grid; c) discretized neuron.

2.1 Formal Definition

Before we discuss the computational strength of the conductive field brain, we need a more rigorous definition. The *conductive field brain* consists of a grid of cells called the *conductive field*, an input, an output, and three types of *neurons*: internal, sensory, and motor. The input consists of sensors and the output consists of motors. Every cell of the conductive field has its own potential and each pair of neighboring cells has defined a conductivity which determines how much potential flows between them every step. There is also a fade-out effect. Every step, a part of the potential is lost in every cell.

An *internal neuron* has two terminations, each connected to a cell, and a threshold. The first termination is called the soma. If the potential in the somatic cell exceeds the threshold, the neuron is active. It is passive otherwise. The second termination is called the axon. A neuron has positive or negative polarity which makes it excitatory or inhibitory respectively. When a neuron is active, it spikes. The spike is a process, during which a neuron diminishes the potential in the somatic cell and changes the potential in the axonal cell. The potential is raised if the neuron is excitatory and it is diminished when the neuron is inhibitory.

The other two types of neurons are derived from the internal neuron. A *sensory neuron* doesn't have a soma. It is connected to a sensor instead and is active when the sensor is active. A *motor neuron* is connected to a motor and doesn't have an axon. When a motor neuron is active, it activates the corresponding motor.

Then the model works in steps. In the beginning, the potential is zero in every cell. A step of a conductive field brain consists of the following actions:

1. input is updated,
2. sensory and motor neurons spike,
3. potential conducts and fades out,
4. motor neurons spike,
5. the output is generated.

We will now build the model formally. Let us consider $h, w \in \mathbb{N}$ to be the height and width of the conductive field respectively.

Definition. *A conductive field with dimensions $h \times w$ is a tuple $F = (P, C, f)$, where:*

- $P \in \mathbb{R}^{h\times w}$ *represents the potentials,*
- $C \in \langle 0, 1\rangle^{h\times w\times 2}$ *stands for the conductance,*
- $f \in \langle 0, 1\rangle$ *is the fade out factor.*

The value $c_{i,j,1}$ determines the conductance of the potential between $p_{i,j}$ and $p_{k,j}$, where $k = (i + 1) \bmod h$. Similarly for the second dimension.

We may have noticed that the conductive field forms a torus. Indeed, the cells at the borders are connected to the cells at the opposite border of the grid. The definitions of the three types of neurons follow. By coordinates we mean the indexes to the conductive field.

Definition. *An internal neuron is a tuple* $N^I = (\mathbf{s}, \mathbf{a}, p, t, d)$*, where*

- $\mathbf{s} \in \mathbb{N}^2$ *denotes the coordinates of the soma,*
- $\mathbf{a} \in \mathbb{N}^2$ *denotes the coordinates of the axon,*
- $p \in \{+, -\}$ *is the polarity of the neuron,*
- $t \in \mathbb{R}$ *stands for the threshold,*
- $d \in \langle 0, 1 \rangle$ *is the spike diminishing factor for the somatic potential.*

A sensory neuron is a tuple $N^S = (i_s, a, p)$*, where:*

- $\mathbf{a} \in \mathbb{N}^2$ *denotes the coordinates of the axon,*
- $p \in \{+, -\}$ *is the polarity of the neuron,*
- $i_s \in \mathbb{N}$ *denotes the sensor index.*

A motor neuron is a tuple $N^M = (i_m, \mathbf{s}, t, d)$*, where:*

- $\mathbf{s} \in \mathbb{N}^2$ *denotes the coordinates of a soma,*
- $t \in \mathbb{R}$ *stands for the threshold,*
- $d \in \langle 0, 1 \rangle$ *is the spike diminishing factor for the somatic potential,*
- $i_m \in \mathbb{N}^2$ *denotes the motor index.*

We sometimes use "a passive inhibitory axon" or similar inexact expression when we talk about properties that belong to neurons as if they belonged to their parts. We are aware of not producing a confusion and we provide the correct expression when needed. The shortened expressions serve to make the text more comprehensible.

Let us now introduce the definition of the conductive field brain.

Definition. *Let* $h, w, s, m \in \mathbb{N}$*. A conductive field brain* B *is a tuple* $(F, I, S, M, \mathbf{s}, \mathbf{m})$*, where*

- $F = (P, C, f)$ *is a conductive field with dimensions* $h \times w$*,*
- I *is a finite set of internal neurons,*
- S *is a finite set of sensory neurons,*
- M *is a finite set of internal neurons,*
- $\mathbf{s} \in \{0, 1\}^s$ *is a vector of sensors,*
- $\mathbf{m} \in \{0, 1\}^m$ *is a vector of motors.*

As we mentioned earlier, the conductive field brain model works in steps. It starts the computation in the time zero with a brain B_0, where the potential in every cell is set to zero. To make a step from time t to time $(t+1)$, we need to possess a brain B_t and an input vector $i_t \in \{0,1\}^s$. Then we can define a set of intermediate step functions $\tau_1, \ldots, \tau_5$ that, when composed, give us the step function τ. We denote the brains that emerge by application of the intermediate step functions by $B_{t,i}$.

- The function τ_1 updates the sensors in B_t to the input values

$$B_{t,1} := \tau_1 (B_t, i_t) = ((P_t, C, f), I, S, M, \mathbf{i_t}, \mathbf{m_t}).$$

- The function τ_2 takes B_t^1 and performs the spikes of the sensory and internal neurons. There are five possible cases that can happen in a cell.

  1. There is no active axon and no active soma in the cell.
  2. There are active somata but there is not any active axon.
  3. The cell contains more active excitatory axons than active inhibitory axons.
  4. The cell contains more active inhibitory axons than active excitatory axons.
  5. The number of active excitatory and active inhibitory axons in the cell is the same and grater than zero.

$$B_{t,2} := \tau_2 \left(B_t^1\right) = ((P_{t,2}, C, f)\, I, S, M, \mathbf{i_t}, \mathbf{m_t}), \text{ where}$$

$$P_{t,2}[x,y] = \begin{cases} P[x,y] & \text{in case 1} \\ P[x,y] \cdot \Pi_{d \in D} d & \text{in case 2, where } D \text{ are the spike} \\ & \text{diminishing factors of the active somata,} \\ 1 & \text{in case 3} \\ 0 & \text{in case 4} \\ 0.5 & \text{in case 5} \end{cases}$$

- The function τ_3 conducts the potential between the neighboring cells

$$B_{t,3} := \tau_3 (B_{t,2}) = ((P_{t,3}, C, f), I, S, M, \mathbf{i_t}, \mathbf{m_t}), \text{ where}$$

$$\begin{aligned} P_{t,3}[y,x] = &(P_{t,2}[(y+1) \ mod\ h, x] - P_{t,2}[y,x]) \cdot \frac{c_{i,j,1}}{5} \\ &+ (P_{t,2}[y,(x+1) \ mod\ w] - P_{t,2}[y,x]) \cdot \frac{c_{i,j,2}}{5} \\ &+ (P_{t,2}[(y-1) \ mod\ h, x] - P_{t,2}[y,x]) \cdot \frac{c_{(i-1) \ mod\ h,j,1}}{5} \\ &+ (P_{t,2}[y,(x-1) \ mod\ w] - P_{t,2}[y,x]) \cdot \frac{c_{i,(j-1) \ mod\ w,2}}{5}. \end{aligned}$$

- The function τ_4 applies the potential fade-out

$$B_{t,4} := \tau_4\left(B_{t,3}\right) = \left(\left(P_{t,3}, C, f\right), I, S, M, \mathbf{i_t}, \mathbf{m_{t+1}}\right), \text{ where}$$

- The function τ_5 performs the spikes of the motor neurons

$$B_{t,5} := \tau_5\left(B_{t,3}\right) = \left(\left(P_{t,5}, C, f\right), I, S, M, \mathbf{i_t}, \mathbf{m_{t+1}}\right), \text{ where}$$

$$m_{(t+1),i} = \begin{cases} 1 & \text{if there exists an active motor neuron connected to the motor } i, \\ 0 & \text{otherwise,} \end{cases}$$

$$P_{t,5}\left[x,y\right] = \begin{cases} P\left[x,y\right] \cdot d_n & \text{if there exists an active motor neuron} \\ & \text{with a soma}\left(y,x\right), \\ P\left[x,y\right] & \text{otherwise.} \end{cases}$$

We may notice an interesting fact. There exists a single step function τ that gets a brain and an input vector as arguments and returns a corresponding brain in the next step, for any correct combination of a brain and input.

Definition. *Let B be a conductive field brain. The step function is a function defined as*

$$\tau(B) = \left(\tau_5 \circ \tau_4 \circ \tau_3 \circ \tau_2 \circ \tau_1\right)\left(B, i_t\right).$$

Lemma 1. *The potential p of every cell of the conductive field is always in the interval $\Lambda = \langle 0, 1\rangle$.*

Proof. The potential is zero in every cell in the beginning of every computation. The spikes of the neurons lead to the values 0, 1, or 0.5 that all belong to Λ. The fade-out factor f is in $\langle 0,1\rangle$ by definition and if $p \in \Lambda$, then $p \cdot f \in \Lambda$. The spike diminishing factor for the somatic potential d also fits into the interval $\langle 0,1\rangle$ and keeps the potential in Λ. Finally during the conduction, every cell gets a new value p based on the potentials of its neighbors $p_1, \ldots, p_4$ and its own current potential p_0 as follows

$$p = p_0 + \sum_{k=1}^{4} \frac{p_k - p_0}{5} = \sum_{k=0}^{4} \frac{p_k}{5} \leq \frac{5}{5} = 1.$$

□

2.2 Computational Strength

To see what the conductive field brain is capable of, we compare it to a concept called an automaton. The automata are whole group of models where the basic is a finite automaton. We consider the definitions used by Sipser [23]. The automata are deeply studied and many relations between various automata models are well known. It is therefore meaningful to compare the conductive field brain to an automaton with similar properties and benefit from the classification that has been already examined.

Our model reads input and writes output every step and stores information using the potentials. The finite state transducer, which is also sometimes called a Mealy machine, can be the first candidate for a comparison. It consists of a control unit and an interface. The control unit is a finite automaton, where every transition from one state to an other yields an output character. The interface includes a reading head on an input tape and a writing head on an output tape. Both the heads are set to the beginnings of the tapes when a computation starts and they move one position forward every step.

There is an other candidate for a comparison. It is called the Turing machine. It has also a control unit and an interface for reading and writing. The interface consists of a single head on a single tape for both reading and writing. The input string is written in the beginning of the tape and is followed by empty characters when the computation starts. The head is positioned on the first character. Every step, the head reads a character, handles it to the control unit and waits for a command consisting of a new character to write and a direction to move. It moves at most by a single character forward or backward. The control unit is again a variation on a finite automaton. It obtains an input character and based on the current state it decides what is the next state, which symbol is written to the tape, and which direction the head goes.

One can notice that we talk about a control unit and an interface in the two descriptions. The nature of the control units does not differ. We can make a pair of copies of the output alphabet for a control unit of a Mealy machine, one of them coding the original character plus move forward while the other stating for the same character and move backward. Then the control units are equivalent. Fore that reason, we state that the difference between the two machines lies in the interface.

In terms of a machine, the conductive field brain is a control unit and needs to be bound to an interface. We connect it to the Turing's interface and call the result a "brain machine". The choice has two reasons. The Turing's interface is a more complex version of the Mealy interface and we show that the conductive field brain can cope with that. Moreover, our model has real numbers in the representation of potential and it is not clear whether the Mealy machine is as strong as the newly created machine. We show that the Turing's model is equivalent to the brain machine. Yet we declare that much of the strength lies in the interface used.

Before we go deeper to the comparison, we discuss one more idea. As it was reasonable for Allan Turing and George Mealy to set their control units into the environment of tapes, it may be reasonable to set the conductive field brain to a different environment. The model is inspired in many aspects by a biological notion of brain. It may be applied to solve problems similar to what the organic counterpart does.

When a brain gets an input, it is typically a representation of a very limited part of projection of an unlimited world. Till it reacts, the world may change. The action corresponding to the output of the brain is then applied to the modified world and produces an other change. Concede the following example to clarify what the situation may look like. A sensory neuron is activated when an animal touches an object. Part of the unlimited real world is a stone and the animal's body approaching it. The representation is a signal in a sensory neuron. Output may be an impulse to a muscle leading to collision avoidance.

An environment meaningful to our model may be a simple robot with a few touch sensors that generate binary input and two motors that steer movement of the robot on a plane. Compared to that, a tape with a reader, writer and a set of characters seem to be sort of a limited world. That kind of world is useful to study some of the properties that a model exhibits. It doesn't answer questions like what happens if the world is dynamic or whether the inputs are sufficient to represent all the relevant information about the reality.

The world we just described fits to the notion of agent systems and is more rigorously defined by Wooldridge [26]. He reuses the environment classification proposed by Russel and Norvig [21]. By that classification, the environment we proposed is inaccessible, non-deterministic, non-episodic, dynamic, and continuous. It means that the agent can't observe whole the environment at a time, the next action does not guarantee what will be its effect, there are not independent episodes where the agent would be evaluated separately, the environment changes itself, not only by the agent's actions, and there is no fixed finite number of variables that would capture the state of the environment. That properties give rise to a complex decision-making problem. On the other hand, the environment, which the Turing machine lives in, is quite an opposite except it is inaccessible.

Since the author believes that the properties of the model should be demonstrated in the simpler world first, we show that what is possible to compute by a Turing machine, that is possible to compute by a conductive field brain machine too. To have a strong basis for the proof, we start with a formal definition of a Turing machine [23].

Definition. *A Turing machine is a tuple* $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$*, where* Q, Σ, Γ *are all finite sets and*

1. Q *is the set of states,*
2. Σ *is the input alphabet not containing the blank symbol* $\sqcup$ *,*
3. Γ *is the tape alphabet, where* $\sqcup \in \Gamma$ *and* $\Sigma \subseteq \Gamma$*,*
4. $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ *is the transition function,*
5. q_0 *is the start state,*
6. $q_{accept} \in Q$ *is the accept state, and*
7. $q_{reject} \in Q$ *is the reject state, where* $q_{reject} \neq q_{accept}$*.*

The Turing machine T *accepts a word* $W \in \Sigma^*$ *if the computation on that word stops in the accept state.*

Let T *accept* W *iff* $W \in M$*, then* M *is recognized by* T *and we denote this fact by* $M = L(T)$*.*

With the Turing machine definition at hand, we can define a machine controlled by the conductive field brain. We rewire the input and output of the control unit to the sensors and motors of the brain. The accept and reject states are represented by accept and reject motors. The start state q_0 is represented by a sensor s_0 that is active only in the first step of the computation. A formal definition follows.

Definition. *A brain machine is a tuple* $(\Sigma, \Gamma, B, m_{in}, m_{out})$*, where*

- Σ *is the input alphabet not containing the blank symbol* $\sqcup$ *,*
- Γ *is the tape alphabet, where* $\sqcup \in \Gamma$ *and* $\Sigma \subseteq \Gamma$*,*
- B *is the conductive field brain*
- $m_{in} \colon \Sigma \to S_{in}$*, where* $S_{in} \subseteq \mathcal{P}(S)$*, is the input map,*
- $m_{out} \colon M_{out} \to Q \times \Gamma \times \{L, R\}$*, where* $M_{out} \subseteq \mathcal{P}(M)$*, is the output map.*

In the rest of this section, we show that for every Turing machine, there is an equal brain machine. The requirement is formulated into the next definition.

Definition. *A brain machine* B *and a Turing machine* T *are equivalent if the equation* $L(B) = L(T)$ *holds.*

Let T be a Turing machine. We then construct a brain machine R that imitates every step of T. For the components of T and R we use the same notation as in the definitions. Let us think of the conductive field of B as of a table. This table represents the transition function δ of the simulated Turing machine. The rows correspond to the states $q \in Q$, except the accept and reject states which are represented by motor outputs. The columns of the table correspond to the input characters $x \in \Sigma$ of the Turing machine. To illustrate this imagination, see Figure 2.2.a.

Every cell of that table is formed by a rectangular part of the conductive field. In particular, it is a grid of 5 columns and 5 rows. We will call this grid $g_{i,j}$, where q_i and x_j correspond to the row-state and column-character notation. We want the grid $g_{i,j}$ to work as the logical *and*. If the row-state q_i and the column-character x_j are both active, it activates the internal and motor neurons according to the transition function $\delta(q_i, x_j)$ of the Turing machine. Let us define a vector of cells

$$a_{i,j} = (g_{i,j}[2,2], g_{i,j}[4,2], g_{i,j}[2,3], g_{i,j}[4,3], g_{i,j}[2,4], g_{i,j}[4,4]) .$$

The notation is depicted in the Figure 2.2.b. We refer to these cells as to the a-cells and to the rest of the cells of $g_{i.j}$ as to the b-cells. We then set the conductance between the a-cells to 0.9999 (strong dotted lines in the Figure 2.2.b). The rest of the conductance values is set to 0.0001. [1]

[1]The reason for the choice of these "nearly one" and "nearly zero" values is strictly practical. We want the core cells c_q, c_m, and c_x to communicate as much as possible and the a-cells serve as an insulator. The values 0 and 1 are banned, because the adaptation algorithm described later can never reach them.

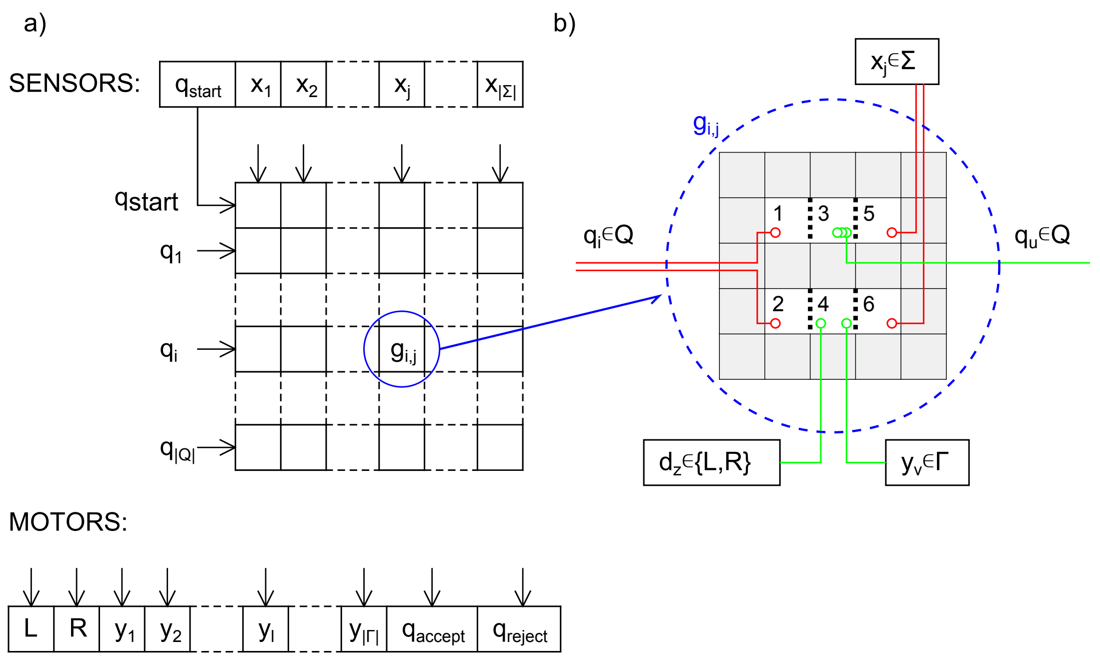


Figure 2.2: Scheme of a brain machine that imitates a turing machine.

As we mentioned earlier, the reading part of the head is mapped the sensors. The brain B has a sensor s_j for each input character $x_j \in \Sigma$. There is a set of sensory neurons for each sensor s_j that connect it to the set of cells in the column j of the table

$$S_j = \{g_i, j, k | i \in |Q|, k \in \{5, 6\}\}.$$

For the start state q_{start}, we have a special sensor and a set of sensory neurons that are active only in the first step. To represent the transition between the states, we make use of the internal neurons. There exists an internal neuron that has its soma in the cell $g_{i,j,3}$ and its axon in the $g_{u,l,r}$, $u \in |Q|$, $x_l \in \Sigma$, $r \in \{1, 2\}$ if and only if

$$\delta(q_i, x_j) \in \{(q_u, y, d) | y \in \Gamma, d \in \{L, R\}\}.$$

We complete the description by defining three types of motor neurons. First of them connects the cells $g_{i,j,4}$ to the motor of the corresponding symbol y. The second type connects the same cell to the motor representing the appropriate direction of head movement d. The last type of the motor neurons connects the cells $c_{i,j,4}$ to the accept or reject motors when $\delta(q_i, x_j)$ leads to accept or reject state respectively. We set the rest of the parameters as follows

- the spike diminishing factors for the somatic potential equal 0.7,
- the thresholds of the motor and internal neurons are set to 0.039,
- the fadeout factor $f = 0.1$.

We now formulate and prove two lemmata which ensure that the grids $g_{i,j}$ work as *and* operands. The first lemma discusses the insulation of the a-cells by the b-cells. The second lemma states that the potential has the right value.

Lemma 2. *The potential in the b-cells is less than* 0.001 *in every step.*

Proof. The potential is zero in the beginning in every cell of the conductive field brain by definition. The b-cells don't contain any axons or somata, therefore the only way how to raise the potential is by the conductance function τ_3. The maximal magnitude of the incoming potential p_{in} is

$$p_{in} \leq 4 \cdot 0.5 \cdot 0.0001 = 0.0002. \tag{2.1}$$

The extreme situation happens if the cell has four spiking neighbors, the fade-out factor is 0.5 and the conductance of the b-cells to all neighbors is 0.0001. Now consider the lowest t, when the cell potential exceeds the bound 0.001 right before the fade-out. Because of the equation 2.1, the potential in the end of the preceding step had to be

$$p_t - p_{in} \geq 0.001 - 0.0002 \geq 0.0008,$$

That was after the fade-out, so right before the fade-out in the preceding step, the potential was

$$p_{t-1} \geq \frac{0.0008}{0.5} = 0.0016 > 0.001.$$

Which is in contradiction to the assumption that t was the lowest. □

Lemma 3. *Let us have indexes $i \in |Q|$ and $j \in |\Sigma|$.*

1. *If there are more active excitatory than inhibitory axons (case 3 of the τ_2 function) in the cells $a_{i,j,1}$ and $a_{i,j,5}$, then the potential p on the cell $a_{i,j,3}$ obeys the inequalities*

$$p \geq 0.039, \tag{2.2}$$
$$p < 0.05 \tag{2.3}$$

   after every step t.

2. *Otherwise the potential p is after every step t bounded by*

$$p < 0.038. \tag{2.4}$$

Proof. We begin with the first inequality $0.39 \leq p$. Let the potentials in the cells $a_{i,j,1}$, $a_{i,j,3}$, and $a_{i,j,5}$ after the spike of the sensory and internal neurons be denoted by $p_{1,a}$, $p_{3,a}$, and $p_{5,a}$ respectively. Recalling that f is the fade-out factor, we get

$$p \geq \left(p_{3,a} + \frac{p_{1,a} - p_{3,a}}{5} \cdot c_a + \frac{p_{5,a} - p_{3,a}}{5} \cdot c_a - \frac{2 \cdot p_{3,a}}{5} \cdot c_b \right) \cdot f.$$

The assumption for the first case of the lemma says that the potentials $p_{1,a}$ and $p_{5,a}$ are right before the conduction equal to one, therefore we can write

$$\begin{aligned} p &\geq \left(p_{3,a} + \frac{c_a (1 - p_{3,a})}{5} + \frac{c_a (1 - p_{3,a})}{5} - \frac{2 c_b p_{3,a}}{5} \right) \cdot f \\ &\geq \frac{p_{3,a} (5 - 2c_a - 2c_b) + 2c_a}{5} \cdot f. \end{aligned}$$

The conductivity between the pairs of a-cells is set to $c_a = 0.9999$ and between the other pairs of cells to $c_b = 0.0001$. Moreover, from the Lemma 1 we know that the potential is always in the interval $\langle 0, 1 \rangle$. For that reason, we obtain

$$p \geq \frac{2c_a f}{5} \geq 0.03996.$$

The potential flow from the cell $a_{i,j,3}$ was bounded from the top for the purpose of the preceding inequality. Let the maximal potential in the b-cells be denoted by p_b. Thanks to the Lemma 2, we know that p_b is less than 0.001 and therefore there is the following bound for the maximal value of p

$$\begin{aligned} p &\leq \left(p_{3,a} + \frac{p_{1,a} - p_{3,a}}{5} \cdot c_a + \frac{p_{5,a} - p_{3,a}}{5} \cdot c_a + \frac{2 \cdot p_b}{5} \cdot c_b \right) \cdot f && (2.5) \\ &\leq \frac{5p_{3,a} + 2c_a (1 - p_{3,a}) + 2c_b (0.001 - p_{3,a})}{5} \cdot f \\ &\leq \frac{p_{3,a} (5 - 2c_a - 2c_b) + 2c_a + 0.002c_b}{5} \cdot f && (2.6) \end{aligned}$$

After substitution of the constants, the bound becomes

$$p \leq 0.06p_{3,a} + 0.01999802.$$

We can now employ the same trick as in the proof of the Lemma 2. The potentials are all zero in the beginning. Consider the lowest time t when p is grater then 0.05. We know that p and $p_{3,a}$ obey the inequality 2.6, therefore the value $p_{3,a}$ is grater than 0.05. Since there is no axon in the cell $a_{i,j,3}$, the potential had to be grater than 0.05 in the end of the preceding step, which is in contradiction to the choice of t.

The last inequality $p < 0.038$ is proved in similar manner to the other two. We know, that at last one of the potentials $p_{1,a}$ or $p_{5,a}$ is not raised by spiking excitatory axons. Lemma 1 says that the potential was at most 1 before the last fade-out, so it is lower than 0.1 after the fade-out. Re-using the inequality 2.5, we obtain

$$\begin{aligned} p &\leq \frac{5p_{3,a} + c_a (1 - p_{3,a}) + c_a (0.1 - p_{3,a}) + 2c_b (0.001 - p_{3,a})}{5} \cdot f \\ &\leq \frac{p_{3,a} (5 - 1.1c_a - 2c_b) + 1.1c_a + 0.002c_b}{5} \cdot f \\ &\leq 0.0779982p_{3,a} + 0.021997804. && (2.7) \end{aligned}$$

This recurrent bound is sufficient to prove $p < 0.038$ assuming that the potential after the preceding step was lower than 0.05. Since there is not any soma of a motor neuron in the cell $a_{i,j,3}$, the inequalities hold till the end of the step. $\square$

Observation 4. *The same inequalities as we proved for the potential of the cell $a_{i,j,3}$ in the Lemma 3 hold for the cell $a_{i,j,4}$ before the motor neurons spike.*

We formulate the theorem about the comparison of computational strength of the brain machine and the Turing machine.

Theorem 5. *For every Turing machine T there exists a brain machine B such that B and T are equivalent.*

Proof. Let T be an arbitrary Turing machine. We construct a brain machine R that imitates every step of T as described earlier in this section. Because the thresholds of the motor and internal neurons are set to 0.039 and because of the Lemma 3, the grids $g_{i,j}$ behave like *and* operators. The potential in the cells $a_{i,j,3}$ represent the state of the machine T and the neurons represent the transition function δ. The brain machine R writes the same symbol to the tape and does the same movement with the head as T does. Because every step of the machines is the same, every computation on the same word $x \in \Sigma$ is the same and B and T are equivalent. $\square$

The other direction of equivalence between the Turing machines and the brain machines is discussed by the following theorem.

Theorem 6. *For every brain machine B there exists a Turing machine T such that T and B are equivalent.*

We don't provide a detailed constructional proof here, because it would be a technical challenge itself. Since the calculation of a brain machine is strictly algorithmic with a finite input and output, there is no expectation that it could outperform the Turing machine in computability. The key idea is that T simulates a computation of B to a limited precision and, if it recognizes that the precision is insufficient at some point of the calculation, it starts again and recomputes the numbers to a higher precision. It can do so since the tape is infinite and the input is stored on the tape. After any finite number of steps, there is a finite set of possible settings that the machine B can reach and therefore only a finite precision is needed to determine the further step (thresholds and potentials can be compared in a finite time).

We also discuss the statement of Siegelmann [22], that there exist a neural network which is computationally stronger than the Turing machine. The key feature that the Siegelmann's model exhibits is the precision of real numbers which enables it to perform infinitely precise computations.

We could think of a modification of the conductive field brain in the way that the thresholds were smooth. That is, the neurons would be equipped with a continuous activation function. However, it raises some non-trivial questions. The current neurons set the potential in the axonal cell to a fixed value if they spike, and they leave the current value if they are inactive. That is completely changed when the activation function is continuous. If we set the potential in the axonal cell to the value of the activation function, the potential might drop even in the case of a spike of an excitatory neuron. If we realized the spike by summation, that is, we added the value of the activation function to the current potential, we wouldn't be able to guarantee the potential values in the interval $\langle 0, 1 \rangle$. Moreover, the signal propagates through a grid where it looses its strength with distance. An infinitely small change in the axonal cell would be much harder to detect than an aggressive setting of the potential to the maximal value.

We conclude that the brain machine, which is base on the conductive field brain, is as strong as the Turing machine. There may exist a stronger variation, but a modification, that would bring this strength to the conductive field brain, is necessarily non-trivial.

3. Searching the space of conductive field brains

In the previous chapters, we introduced and defined the conductive field brain and the brain machine that incorporates it. We showed that for every Turing machine, there is a configuration of the model that behaves exactly the same. Existence of such a structure motivates us to search for the right setting for a given problem.

We divide the search for the right conductive field brain into two parts. General properties have to be determined first. This includes the size of the grid, count of sensors, motors, and number of neurons and relative positions of their somata and axons. Secondly we need to set the conductance between neighboring cells of the conductive field and thresholds for neural activation. We consider the first group of the properties to be stable and not changeable during the life of the brain. There is evidence that these properties change in a human brain. We have decided to neglect these properties in the sake of simplicity. The second group of the properties is conceived to be much more dynamic and is adapted while the conductive field reacts to the environment.

3.1 Signal interpretation

The search for the right setting of the model is a problem-dependant task. We need to feed the right input and interpret what happens to obtain an output. Lets examine the execution of the the model from a larger scale of view for that reason. The conductive field brain takes a vector of boolean values as an input and gives an other boolean vector as an output each step. The model does not exhibit any implicit meaning of the vectors and it does not say what is the relation between a set of consecutive inputs or outputs.

Encoding of information in an organic brain is a subject of intensive research. Spiking frequencies of neurons seem to carry a lot of information, though there is evidence suggesting that the temporal properties of neural activity are rather more complex [20]. The true firing rate of a set of neurons is unknown and its estimation isn't a simple task [25]. However, ae find inspiration in the spiking frequency encoding when interpreting the input and output signals of the model. Our approach is simplified to the extent that we encode intensity of a sensory stimulus into spiking frequency of sensory neurons. To obtain an output, we measure the firing rate of motor neurons. While the other temporal properties of the signal are neglected in the view from outside, they may carry some additional information inside the model.

3.2 Conductance

We now set our focus to the conductance adaptation that models the dendrite modification. The source of motivation and inspiration comes again from biological discoveries. A lot of evidence indicating that the dendritic growth is highly dynamic has been described during the last two decades [13], [28]. Among the other factors, the dendrites seem to be reshaped by neuronal activity. Little is known about what role the modifications play in memory formation and what are the key principles which determine the future shape of a dendrite [12].

Biological observations suggest that the dendrite shaping process is modulated by selective attention [18]. The changes are made to the parts of the brain, that is responsible for the decision being made. We cope with that by saying that our model shall be trained on a single problem and therefore there is no need to modulate the learning process selectively by attention. In other words we model a part of a brain tissue that pays attention as a whole. On the other hand, Friston [10] proposed that the reward signal in the brain is not targeted to particular units, but is rather distributed in a generalised way. This suggestion was supported by a strong evidence observed by FitzGerald [9] using the fMRI. If we accept that the credit assignment problem is not solved entirely by the reward distribution mechanism, then we need to figure out in what way is processed the local information.

We address this issue by the following idea. Suppose, that a neuronal soma receives a reward signal. The neuronal body can store information about intensity of its last activity and it could conclude from which part of the soma or a dendrite the input signal came. Combination of these factors could drive the dendrite modification process. A dendrite could then strengthen in the direction of the input signal that precedes the reward. The information needed could be stored somewhere in the chemical properties of the neuronal membrane for example. The author modestly admits that this idea is solely a product of his imagination and that he has not any supporting biological facts except the sources cited. Instead, we hereby present a model and its computer implementation in the following chapter and deliver some supportive experimental results.

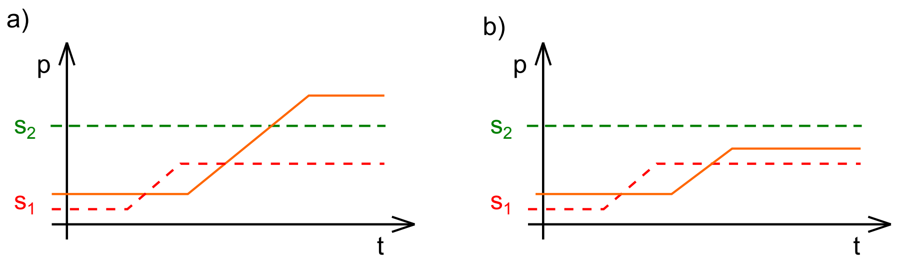
As a consequence, we obtain a mechanism that is close to the principle known as Hebbian learning. Hebb's idea was that two neurons which are simultaneously active should develop a degree of interaction higher than those neurons whose activities are uncorrelated [19]. Our approach is more detailed and it incorporates the feedback information. If we supplied our model with a steady reward, we would expect the Hebbian behavior to emerge. The model called Hopfield networks is known to implement a variant of the Hebbian learning [19]. However, the Hopfield networks apply the idea as is, while we state that such a behavior may emerge in our model from a set of lower-order rules.

In principle, a reinforcement learning method is used. The adapted model reacts to an environment. It produces output and gets back some feedback from the environment. The model may change its properties based on the feedback and then it produces next output to get an other feedback.

3.3 Thresholds

The internal and motor neurons need their thresholds to be set. The task is mainly to tune the thresholds so they split the strength of potential that can occur at the place of the neural soma. The author doesn't have any essential clue of what the right mechanism is. We therefore use the simplest working mechanism at hand. It is a result of trial and error method and an additional

Figure 3.1: *Threshold tuning for multiple levels of signal strength.* The threshold is painted in orange solid line when a) bounds are applied to the threshold value b) bounds are applied to the activity value. Potential on the somatic cell is represented by the vertical axis and the horizontal axis states for time. The red dashed lines (s_1) describe the potential level in the situation when high activity is expected and the green dashed lines (s_2) depict the situation when low activity is expected.



condition. We require that the mechanism is empathic, which means that it uses only information accessible at place and it doesn't demand any complex strategy or global knowledge.

A neuron monitors change of two quantities, the reward and its own activity. If the activity changes in a certain way and the reward raises at the same time, the threshold moves to support the activity change. The threshold drops in case of raising activity and vice versa. When the reward drops, the threshold value moves in the opposite way and it continues in the movement by the last decision in case of stable reward or activity. The mechanism is illustrated in the Figure 3.1. This approach brings two problems to solve.

The first is an additional parameter, because we have to know how much the threshold should rise or drop. If the step is too small, the algorithm is too slow. If we make the step too big, the precision is lost and it tends reach unreasonably big numbers. We therefore set a relatively large step in the beginning and we make it smaller every time the desired direction of the threshold modification changes. The size of the step in the beginning brings an other problem. It can rise too quickly so that the change of activity of the neuron cannot be detected. In that case the rule of continuing in the last direction is applied and the threshold value diverges. For that reason, the bounds for lowest and highest activity are set. If the activity approaches zero or it is hundred percent for some time (active in every step) the threshold movement stops.

We may ask a question why the bounds are set to the activity instead of the threshold. The reason is that there may be multiple levels of signal strength in different situations. A schematic example is depicted in the Figure 3.1. When a threshold is low and the somatic potential gets a little higher, activity rises. If a negative feedback comes, the threshold starts to rise. It may be perfectly fine that the activity becomes zero in that situation. If the threshold was bounded, it would rise till the bound. But there comes the second case, when the potential is higher than before and the desired activity is high. For that reason, we stop the potential change when the activity reaches an extreme value. Than there is a chance that the threshold stops in between and separates the two cases.

3.4 Evolution

We have described adaptation of conductance and thresholds in the previous sections. These methods adapt a model with a given *architecture*, that is, when size of the conductive field, somata positions, and routing of axons are fixed. The biological counterparts of these properties are generally considered to be genetically predetermined to a high extent [2]. For that reason, we employ an evolutionary algorithm (EA) to search the space of available architectures. The implemented algorithms are presented and their resemblance to some well-known variants of EA is pointed out.

There are many variants of evolutionary algorithms, though most of them share the same underlying idea. They work with a population of individual solutions of a given problem. The environmental pressure chooses the fittest candidates to survive and to seed the next generation. According to Eiben and Smith [8], the following components of an evolutionary algorithm need to be defined in order to specify particular evolutionary algorithm

- representation (definition of individuals),
- evaluation function (or fitness function),
- population,
- parent selection mechanism,
- variation operators, recombination and mutation,
- survivor selection mechanism (replacement).

First, we discuss what an *individual* is. A concept, that needs to be understood, is that an individual is a solution candidate in the context of the problem. It is the actual architecture of a conductive field brain. This individual is represented by a *genome* in the context of the EA. There may be multiple genomes that represent the same architecture, but only a single architecture exists for every genome.

The encoding of individual architectures into genomes influences efficiency of the algorithm. The tasks we want it to solve are

- choose the right size of the grid,
- determine the counts of internal, sensory and motor neurons,
- position the neurons each to other,
- determine polarity of the neurons,
- select the right sources and targets for the sensory and motor neurons resp.

The number of motors and sensors is given by the definition of the problem. The information is stored along with the genome but it is not expected to evolve. We choose the following encoding for the rest of the parameters. The size of the grid is represented by two integers. The neurons are represented by a chain of *genes*, each stands for a single neuron. A gene contains a pair of integers representing coordinates of a soma and an axon (or an index of a motor/sensor in case of motor or sensory neuron) and a boolean flag which determines whether the neuron is excitatory or inhibitory. More discussion on the topic is provided later in this section and an illustrative example can be found in the Subsection 4.1.1.

The evaluation of individuals is given by a particular application of the model and is discussed separately. Examples of the fitness function may be found in the Chapter 5 in the experiments description. For the purposes of this section, we conceive it to be a function which takes an individual and gives a real number between -1 and 0, the higher the number, the fitter the individual.

Let us now discuss the mutation operator. The problem, whether mutations are purely random or they are directed, has been an open question since Charles Darwin came with his famous theory of evolution [17]. Consider an example of bacteria resistant to certain antibiotics for illustration. Those, who argue for the randomness, state that the mutation of bacteria which caused the resistance would occur, even if people were not using the antibiotics. The resistance wouldn't be of any advantage and it might disappear, but the mutation would occur. It means, that the mutation would have happened before the antibiotics tried to kill the bacteria concerned.

Because our aim is to create a genome with certain properties, we may benefit from a directed mutation that uses additional information gathered during the genome evaluation. As mentioned earlier in this chapter, higher conductance between a somatic cell and its neighboring cells should develop when the signal coming from the particular direction is more important. Following this construct, the additional information which directs the mutation is a grater probability of neuron movement in the direction of higher conductance. We will use this directed mutation (i.e. movement of the soma in the direction of highest conductance) as an alternative to a purely random mutation.

The population doesn't have any specific structure in our EA. It is represented by a list of genomes with a fixed size. However, we need to describe how it develops in time. We provide several versions of the algorithm based on the complexity of the problem.

First, we describe the simplest approach which we call *guesser*. It is inspired by the variant of the EAs called the *evolutionary strategies* [8]. However, we do not try to imitate that algorithm. As the name suggests, our algorithm mostly tries to "guess" the right genome and then refine it by mutation. Guesser is a special case of EA where populations contain only a single genome. It loops through consecutive generations and it either generates a new genome or mutates the preceding one. Guesser takes a parameter $BlindMuts$ which limits the longest sequence of the consecutive mutations. When this limit is broken, a new genome is generated. Guesser returns the fittest genome over all generations. A scheme of the guesser algorithm is depicted in the Algorithm 1. It is designed to solve those elementary problems which are hard to decompose and where the solution is typically small. The selective pressure is not of much use and a lot of exploration

Algorithm 1 Scheme of the guesser algorithm.

```
procedure GUESSER($MaxTrials$, $BlindMuts$)
    $t \leftarrow 1$
    *Initialize* a new genome $G_0$
    *Evaluate* genome $G_0$
    $Best \leftarrow G_0$
    while $t <= MaxTrials$ AND $Best$ is not satisfactory do
        if $Best$ has not been changed in the last $BlindMuts$ loops then
            *Initialize* a new genome $G_{t+1}$
        else
            $G_t \leftarrow$ *Directed mutation* of genome $G_{t-1}$
        end if
        *Evaluate* genome $G_t$
        if $G_t$ is fitter than $Best$ then
            $Best \leftarrow G_t$
        end if
        $t \leftarrow t + 1$
    end while
    Return $Best$
end procedure
```

is needed.

Next we present an algorithm which we call the *evolution* for the purposes of this work. It takes three parameters *MaxGenerations*, *CrossoverFlag*, and *EliteSize*. The first limits the count of generations to guarantee that the algorithm stops. The second parameter determines whether a crossover is used to produce a successive generation or only mutant *survivor* genomes selected from the current generation are used. The parameter *EliteSize* settles the count of the fittest genomes which pass to the next generation mutated with the directed mutation.

The algorithm loops through the generations. Every time it passes the elite to the new generation. Then the algorithm completes the new generation, either by offspring of selected parents or by selected survivor genomes after random mutation. A scheme of the evolution algorithm is depicted in the Algorithm 2. Both types of mutation are used in the evolution algorithm. The survivor genomes are mutated randomly to keep the diversity of the new generation. In fact, there wouldn't be any difference between elite genomes and survivors if the directed mutation was used. When a new offspring genomes are produced, the random mutation is used too. There is a pragmatic reason in that case. The directed mutation needs fitness to be computed.

The *roulette selection* is used when parents are chosen to breed or genomes are selected to survive. The genomes are evaluated by fitness function with values in the interval $\langle -1, 0\rangle$. A number 2 is added to the fitness of each genome to obtain positive values. These values then represent lengths of intervals. The first interval starts at zero and every successive interval continues were the predecessor ended. We then choose a genome by generating a random number in the union of the intervals. That genome is selected which interval contains the random number.

Algorithm 2 Scheme of the evolution algorithm.

```
procedure EVOLUTION($MaxGenerations$, $CrossoverFlag$, $EliteSize$)
    Initialize a population $P_0$
    Evaluate every individual in $P_0$
    Best $\leftarrow$ the fittest individual in $P_0$
    $t \leftarrow 0$
    while $t < MaxGenerations$ AND Best is not satisfactory do
        Initialize an empty population $P_{t+1}$
        Clone EliteSize fittest individuals from $P_t$ to $P_{t+1}$
        Apply Directed mutation to all individuals in $P_{t+1}$
        if $CrossoverFlag$ then
            Select parents using Roulette selection
            Breed the parent pairs using Crossover
            Mutate the offspring using Random mutation
            Append the mutated offspring to $P_{t+1}$
        else
            Select survivors using Roulette selection
            Apply Random mutation to survivors
            Append the mutated survivors to $P_{t+1}$
        end if
        Evaluate every individual in $P_{t+1}$
        Best $\leftarrow$ the fittest individual in $P_{t+1}$
        $t \leftarrow t + 1$
    end while
end procedure
```

The last component from the Eiben and Smith's list is the recombination realised by the crossover in our evolution algorithm. Let us begin the discussion with motivation. One of the variants of evolutionary algorithms are the *genetic algorithms* presented by Holland [15]. Part of the theory is the *building blocks hypothesis*. It says that if there exist low-order patterns with above average fitness in the genome, than they are favoured in the selection. This way the better building blocks are preserved and recombined in the subsequent generations to compose a genome with high fitness. One can imagine that the building blocks can concretize into patterns of relative placement of the neurons. Modules with specific function may then emerge in the population.

The neuron gene chain is coded with respect to that idea. The first neuron in the genome is placed arbitrarily. Every consecutive neuron codes its position relatively to its predecessor. Different positions of the first neuron therefore produce symmetrical solutions and therefore the first neuron can be always placed to the position $[0, 0]$. We expect it to have the following effect. A substring of the neuron genes, that is transferred from one genome to an other, keeps its relative positioning. Moreover, it is placed relatively to its preceding neuron. We therefore inspire in the *single-point crossover* proposed along with the genetic algorithms. In short, we cut the chains of parental neuron genes at a random position and take one part from each to produce the child neuron gene chain. To recombine the numeric parameters, we again inspire in the evolutionary strategies. For every parameter, the descendant obtains a random value from uniform distribution between the parental values of the parameter.

We don't examine the mechanism based on the building blocks hypothesis experimentally. We rather focus on the basic properties of the conductive field brain where the simpler algorithms (guesser and evolution without crossover) are satisfactory and modules don't bring any advantage. We provide the evolution algorithm with crossover to show what effects the spatial nature of the model can bring. In addition, we suggest one more concept for future work. The guesser and evolution algorithms may work together in a combination. The genomes produced when solving elementary problems using guesser may be used as starting building blocks for creation of new genomes in the evolution.

4. Implementation

We have proposed a new computational model in the previous chapters. It was examined theoretically and heuristic methods of searching for a relevant genome and adaptation of a conductive field brain were described. However, the description is sometimes provided in a high level of abstraction. In this section, we concretize those issues that were not discussed in detail and we present a software implementation.

We begin with a depiction of general structure of the solution. Then we describe each of its parts in more detail. We concretize the method of adaptation as well as interpretation of the input and output signal with respect to time. Since it is necessary to implement an environment for every new experiment, the application was created with an intention of further development. We therefore demonstrate how a simple experiment is added to the program [1] in the Appendix A. An instruction manual of how to run the program can be found int the Appendix B.

The model is implemented in the C# language in Microsoft .NET Framework 4. It was developed and tested on the Microsoft Windows 7 operating system. The application was coded using Microsoft Visual Studio and a solution with the application code forms a part of this work. The solution is divided into a library called `CoFiBa` and an application `CoFiBa_App`. The library implements logic of the model, while the application reads and writes to files and it contains the classes derived from the class `World` which specify particular experiments. The key classes with their essential fields and methods are depicted in the Figures 4.1 and 4.2.

4.1 CoFiBa library

The `CoFiBa` library is organised into four thematic namespaces. Three of them, `Genome`, `Brain`, and `Environment`, are labeled in grey in the Figure 4.1 and we will describe them in the same order they are listed. The fourth namespace, called `Utils`, contains universal classes and methods that are used across all the other namespaces or that do not fit to any of them. We will refer to `Utils` in various places of the text.

One of the `Utils` classes is the static class `Settings`, which contains default values and constants used in the solution. It is divided into four regions, each of them contains settings related to a particular topic. The settings are listed in the Table 4.1. An other static class serving as a container is the class `Functions`. As the name suggests, it encapsulates various methods that don't fit elsewhere. We give the `Mod` function which defines modulus for both positive and negative integers or the `RandInt` function that populates the static common pseudo-random generator `Next` method as examples.

[1] Good knowledge of the programming environment of C# .NET is a prerequisite.

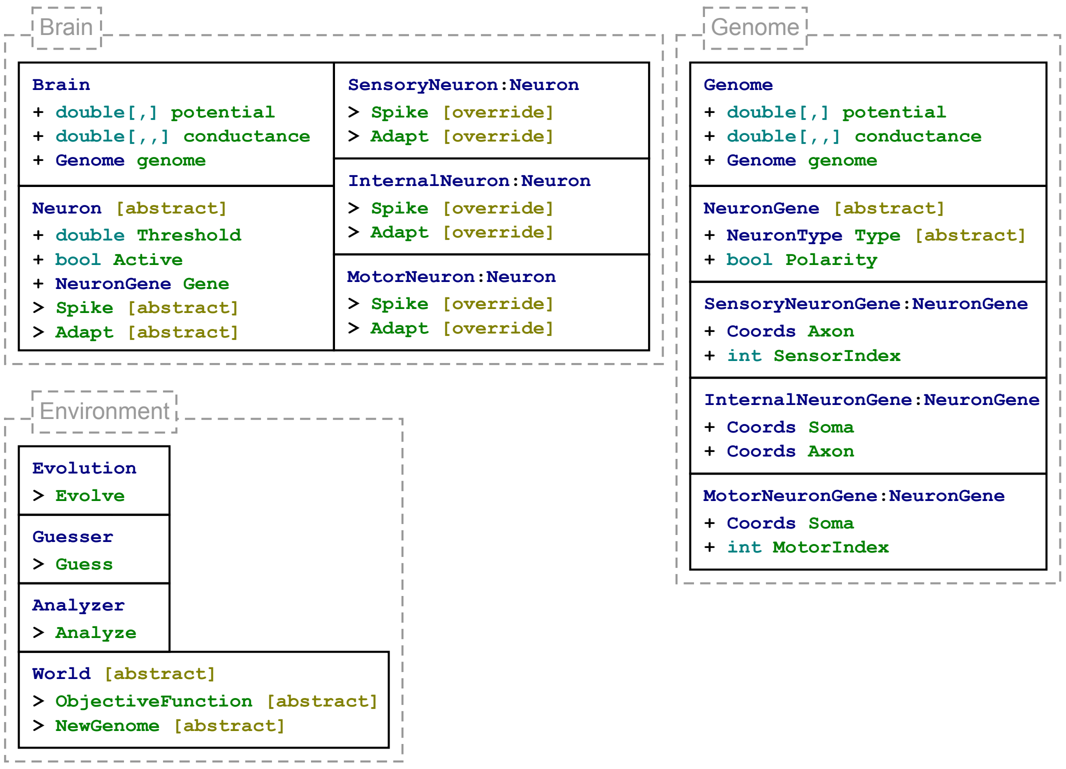


Figure 4.1: classes `CoFiBa` library

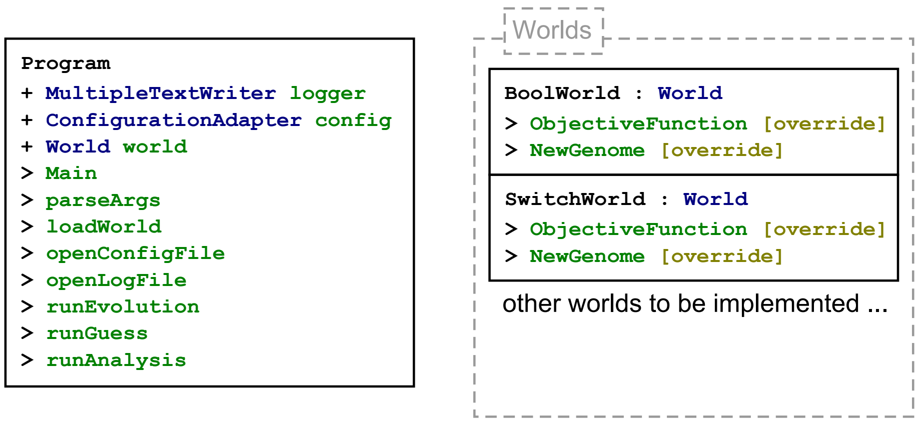


Figure 4.2: classes application `CoFiBa_App`

Conductive field	
Name	Description
CF_pfo	Potential fade-out.
CF_rfo	Conductance fade-out.
CF_maxConductance	Maximal conductance.
CF_defConductance	Default conductance.
CF_defPotential	Default potential value.
CF_CmodDimFac	Conductance modulator diminishing factor.

Neurons	
Name	Description
Neur_Inhib	Probability that a newly created neuron is inhibitory (else it's excitatory).
Neur_Threshold	The default activation potential of a neuron.
Neur_Momentum	The activity momentum.
Neur_WeakSignal	What level of signal is considered to be weak.
Neur_SpikeSrcFac	How much is the potential on the soma diminished when a neuron spikes.
Neur_ChIntImplMom	Imploding momentum of a change interval (used for activity and feedback change metric).
Neur_ChIntExplMom	Exploding momentum of a change interval (used for activity and feedback change metric).
Neur_ThrChMag	The initial magnitude of threshold change.
Neur_ThrChMag2	Change of change of threshold magnitude.

Mutation	
Name	Description
Mut_AppSenFree	Chance of appearing a new sensory neuron when a sensor input is not occupied.
Mut_DisappMultiSens	Chance of a sensory neuron removal when more than one sensory neurons are connected to a sensor.
Mut_AppMotFree	Chance of appearing a new motor neuron when a motor output is not occupied.
Mut_MovNeuron	Chance of moving a neuron.
Mut_RemoveNeuron	Chance of removing a neuron.
Mut_AddNeuron	Chance of adding a neuron.
Mut_PolarChange	Chance of a change of a neuron polarity.

Table 4.1: Constants and default values

4.1.1 Genome

The purpose of the classes included in the namespace `Genome` is to store and perform operations on the conductive field brain genome. It is depicted in the right of the classes scheme in the Figure 4.1. The class `Genome` contains

- size of the conductive field,
- number of sensors and numbers of motors,
- list of neuron genes.

Several operations can be performed with a genome. Two constructors are defined that create a new genome either as an empty shell without any neuron genes or with a specified number of randomly generated genes. There are two other methods that return a genome. The method `Clone` creates an exact shallow copy. The method `Bride` takes two genomes as arguments and returns a new one that results from crossover of the two and applies a subsequent mutation.

The crossover is a mixture of ideas from multiple resources. It recombines the numeral parameters in a similar fashion to what Storn and Price suggested in the algorithm called the differential evolution [24]. In addition to that, the strings of neurons are cut at a random place and the resulting string takes a substring preceding the cut from one parent and the substring following the cut from the other. It is an analogous method to what was introduced by Holland [15].

The method `Mutate` of the `Genome` class changes the genome. It takes two boolean arguments. The argument `directed` determines whether the mutation is purely random or it is directed by the knowledge gathered when running the objective function as discussed in the Section 3.4. The second argument is called `stableNeurCounts` and it indicates whether the neuron string should be mutated (i.e. neurons added or removed). All variants of the mutation propagate the activity to the neurons included in the genome.

The neurons are represented by three classes `SensoryNeuronGene`, `InternalNeuronGene`, and `MotorNeuronGene`. Each of the three types contains coordinates of a soma and of an axon where applicable. The motor and sensory neuron genes also contain index of the sensor or motor they are connected to. All of them inherit from a common class `NeuronGene`. This class populates abstract methods `Mutate` and `Clone` that are shared by all three types. They are called when the class `Genome` propagates the call of methods with the same name. The neuron genes also encapsulate the polarity parameter of the neuron.

The mutation uses a pseudo-random generator to modify the polarity parameter and to move the axon or soma. This is where the directed mutation differs. Probabilities of move of a soma or of an axon to the four neighboring cells are weighted by the corresponding conductance values. The classes representing the particular types of neurons override the `ToString` method and add a static `FromString` method. This couple of functions is used for serialization of the genome that enables the user to save and load the genome from a file.

We now describe the format of a serialized genome. Whole genome can be formatted into a single row (i.e. not containing the CR and LF characters). One of the advantages is that we don't need any special treatment for genomes as parameters of a configuration file that is processed row by row. An example of a genome follows. It is structured to multiple lines for a human reader, but we can imagine that all the newline characters are replaced by a space character in the configuration file.

```
Genome(S:2 M: 1 [13, 8]):
  +Sensory(Axon[7, 7] SensorIndex:1 Polar: Excit)
  +Sensory(Axon[12, 1] SensorIndex:0 Polar: Inhib)
  +Internal(Soma[9, 1] Axon[3, 5] Polar: Excit)
  +Internal(Soma[4, 0] Axon[8, 7] Polar: Excit)
  +Motor(Soma[0, 2] MotorIndex: 0 Polar: Excit)
```

All the numbers are variable and the code is not case sensitive. The string starts with the keyword "Genome" followed by properties in round brackets in a fixed order

1. number of sensors, preceded by "S" and a colon,
2. number of motors, announced by "M" and a colon,
3. width and height in a pair of square brackets.

Then a colon introduces the list of neurons. Every neuron definition starts with a "+" sign and a type of the neuron. Parameters of the neuron are enclosed in a pair of round brackets. `Sensory` neuron contains the following parameters in a fixed order

1. the coordinates of the axon enclosed in squared brackets and preceded by the keyword "Axon",
2. "SensorIndex:" followed by the index of the sensory input that determines activity of the neuron, and
3. polarity "Polar:", that can be "Excit" for excitatory neuton or "Inhib" for inhibitory.

The genes for `Motor` and `Internal` neurons are analogous as apparent from the example.

4.1.2 Brain

The conductive field brain model is represented by the class `Brain`. The containing namespace is depicted in the Figure 4.1. To create a new brain, one needs a genome. Then a constructor can be called which initializes a new brain with the default potential, conductance and thresholds. The default values are specified in the `Utils.Settings` class which is described by the Table 4.1. Besides some technical data structures, the brain contains its genome, a list of neurons, sensors, motors, and the following arrays

- `potential` keeps the values of potential in the cells of the conductive field,
- `conductance` stands for the conductance between the neighboring cells,

- `pmov` represents the magnitude of the potential flow between the neighboring cells,
- `cstren` and `cstreni` represent the conductance strengthening signal[2] for excitation and inhibition respectively.

The sensors and motors are represented by objects of the class `Signal` defined in the `Utils` namespace. `Signal` is initialized to work in one of two modes. It can play a role of an emitter or a receiver. The former gets strength of the signal as a parameter and it produces a series of boolean values indicating whether to spike or not. It is used in the case of sensors. The latter counts the number of steps between consecutive spikes and updates the detected signal strength accordingly. Motors are represented by receivers.

The brain performs two basic actions. It makes a step by the `Step` method and it adapts by the `Adapt` method. The `Step` method implements the step function defined in the Section 2.1. Since the implementation follows the definition, we don't describe it further. The `Adapt` method needs to me examined deeper. Although the general idea is described in the preceding chapter, there are some more non-trivial concepts in the implementation.

Before we present the adaptation algorithm in detail, we describe the classes representing neurons. As in the case of the genome, there is an abstract class `Neuron` which contains the parts of code that are common for all three types of neurons. Each neuron stores the corresponding gene and populates its properties. It also encapsulates the threshold and a `Signal` object. The `Signal` is initialized to be in a receiver mode and it analyzes strength of the signal emitted by the neuron. The brain calls the function `CheckSpike` of every neuron in every step. The neuron compares the somatic potential to the threshold and decides whether to spike or not.

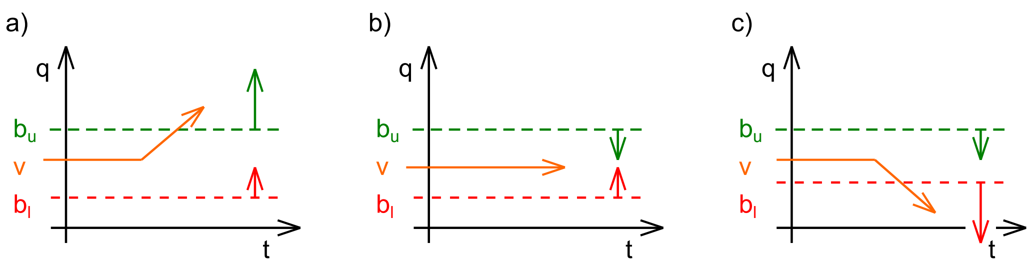
After the brain performs a step, it may adapt. The principal value which determines the adaptation is the feedback. The feedback is set to zero in case of ideal behavior of the brain and scales to one when the output is totally wrong. The adaptation changes two kinds of quantities. The neurons adapt their thresholds as described in the Section 3.3 and the conductance is modified.

The conductance adaptation also starts in the neurons. They keep a simple statistic of the preceding activity and the feedback in the form of an interval. Bounds of the interval move with a momentum towards the current value. The upper bound raises faster and drops slower. The lower bound obeys symmetric rules. When a value exceeds the upper bound, it is considered to be growing and if it descends bellow the lower bound it is considered to be dropping. This logic is implemented in the class `ChangeInterval` and the key situations are depicted in the Figure 4.3.

If the feedback grows, then every neuron emits one of two conductance modification signals based on its activity change. When the activity raises, it strengthens the excitatory pathways. The inhibitory pathways are strengthened when the activity drops. Nothing happens when the feedback doesn't grow. The conductance strengthening signal is represented by the arrays `cstren` for excitatory

[2]The signal is emitted by neurons when certain conditions are met. The algorithm is described later in this section.

Figure 4.3: *Three cases of the ChangeInterval update.* The interval is updated with a value a) grater than the upper bound, b) between the bounds, c) lower than the lower bound. The upper and lower bounds are represented by a green or red dashed line respectively. The horizontal axis stands for time while the vertical for the underlying quantity.



and `cstreni` for inhibitory pathways. The conductance excitatory and inhibitory strengthening signal is diffused each step against or in the direction of the strongest flow of the potential respectively. The signal strengthens the conductance while moving to the neighboring cells and part of it disappears by every step.

We have ignored the object `Analysis` of the class `AnalysisSeries` till now. It is a technical feature, that records selected values during the computation of the brain. These time series can be later used in the `Analysis` mode. The feature is turned on or off by the constructor parameter `analyze`.

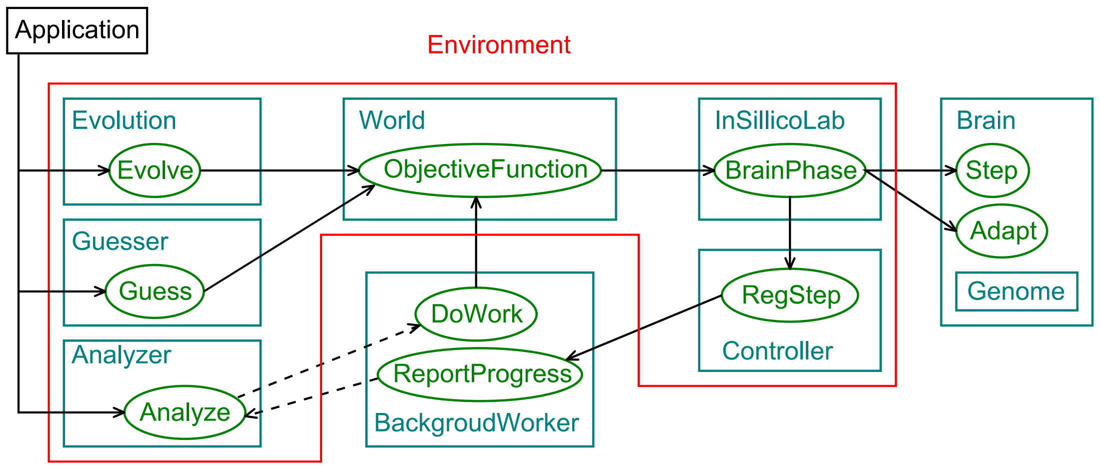
4.1.3 Environment

The third namespace is called `Environment`. It consists of classes that build a context in which the brain exists. The logic of the function calls is illustrated in the Figure 4.4. There are three classes that represent different modes in which the brain can be cultivated. By cultivation, we mean the algorithms which determine when a new genome is created, mutation or crossover occurs and which call the world's objective function to evaluate genomes. The class `Evolution` realizes the algorithm that is described as Evolution in the Section 3.4. The Guesser algorithm from the Section 3.4 is implemented in the class `Guesser`.

While the purpose of the first two classes is to find the right genome for a given problem, the class `Analyzer` takes a genome as an argument. It provides a graphic user interface and brings some insight into the work of the conductive field brain. As depicted in the Figure 4.4, the objective function is called asynchronously to enable user input during the computation. There is also the class `Controller` which serves as a communicator between the user interface and an asynchronous thread that executes the objective function.

We have already mentioned the abstract class `World` which encapsulates the objective function. The objective function is left to be implemented in the application. The idea behind is that the `CoFiBa` library serves as a reusable set of tools where the model is implemented. A researcher who wants to work with the model does not need to re-implement it, because it is sufficient to add a new class which implements the abstract class `World` and benefit from the rest of the

Figure 4.4: *Classes are depicted by blue boxes and methods by green ellipses. An arrow symbolizes a function call (they are directed from the caller to the called). A dashed line of arrow stands for an asynchronous call. The environment namespace is painted in red.*



application.

4.2 The Application

The `CoFiBa_App` project is composed of a class `Program` containing the entry point to the application and a namespace `Worlds` where particular experiments are implemented. In general, the program does four steps. In every run it

1. parses the command line arguments (configuration and log file paths),
2. initializes a world where the brains live,
3. runs a cultivator (`Guesser`/`Evolution`/`Analyzer`),
4. closes the log file.

The first and the last step are technicalities, yet we it find important to point them out. The log and configuration files are loaded in the very beginning, when the program processes command line arguments, and the log is closed in the end of the program execution. This enables the two steps in the middle to read the configuration and to log during all the time of their execution.

The second step initializes a class that inherits from the `World` abstract class. The configuration of the particular experiment is processed during that step. The third step is where the work is done. One of the three cultivators is called. When it is finished, the program closes the output streams and quits.

5. Experiments

In this chapter, we present several experiments to demonstrate some of the properties of the conductive field brain model. The key characteristics of the adaptation method are studied in the *SimpleWorld* experiment in the Section 5.1. Next, we show that the conductive field brain is able to play a role of an activity pattern generator. This pattern generator can be turned on or off by a short impulse to one of the sensors. Such a generator is developed in the experiment *SwitchWorld* which described in the Section 5.2. The experiment *BinBoolWorld* summarized in the Section 5.3 shows the the ability of the model to solve elementary logical tasks. And finally the experiment *UniBoolWorld* presents an example of a genome gained by evolution that adapts to two different problems in the Section 5.4.

We run the experiments on a computer equipped with Intel Core 2 Duo CPU running at frequency 2.66 GHz and 4 GB of RAM. Most of the experiments are executed within a minute. The execution times are discussed in the Section 5.3 in more detail.

5.1 SimpleWorld

In the first experiment, we demonstrate two key properties of the adaptation mechanism of the conductive field brain. The first is the ability to adapt the conductance based on the feedback and the input signal. The second is the way how the thresholds are tuned using more and more precise steps.

For that reason, there is implemented a simple world with two inputs (sensors) and two outputs (motors). The only task of the model is to assign the right output to the right input and to activate the output when the input is active. We don't employ the evolutionary search in this example. Instead, we provide a static genome[1].

```
Genome(S:2 M:2 [11, 12]):
 +Sensory(Axon[2, 6] SensorIndex:0 Polar: Excit)
 +Motor(Soma[3, -4] MotorIndex: 0 Polar: Excit)
 +Sensory(Axon[3, 4] SensorIndex:1 Polar: Excit)
 +Motor(Soma[-3, 3] MotorIndex: 1 Polar: Excit)
```

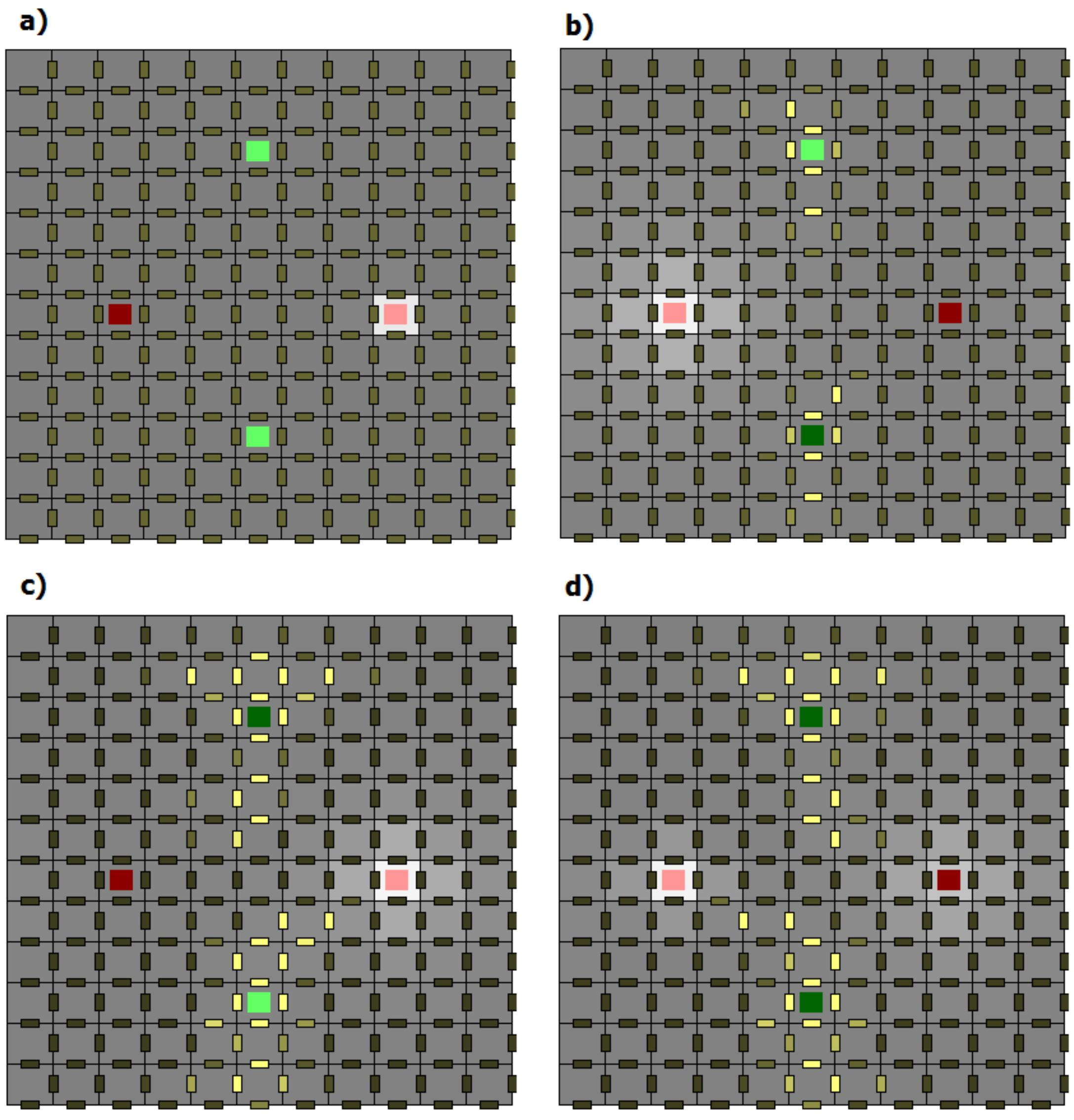
The conductive field brain contains two sensory and two motor neurons connected to the two sensors and two motors respectively. The configuration of the model is illustrated in the Figure 5.1. The neurons are organised clockwise in the same order as they appear in the definition, beginning on the left. Notice that there is a left-right symmetry in the genome. The distance from any of the two motor neurons to both sensory neurons is the same. This enables us to demonstrate how the conductive field brain copes with various feedback.

There are two input vectors $\mathbf{i^0}$, $\mathbf{i^1}$, and two output vectors $\mathbf{e^0}$, $\mathbf{e^1}$, where

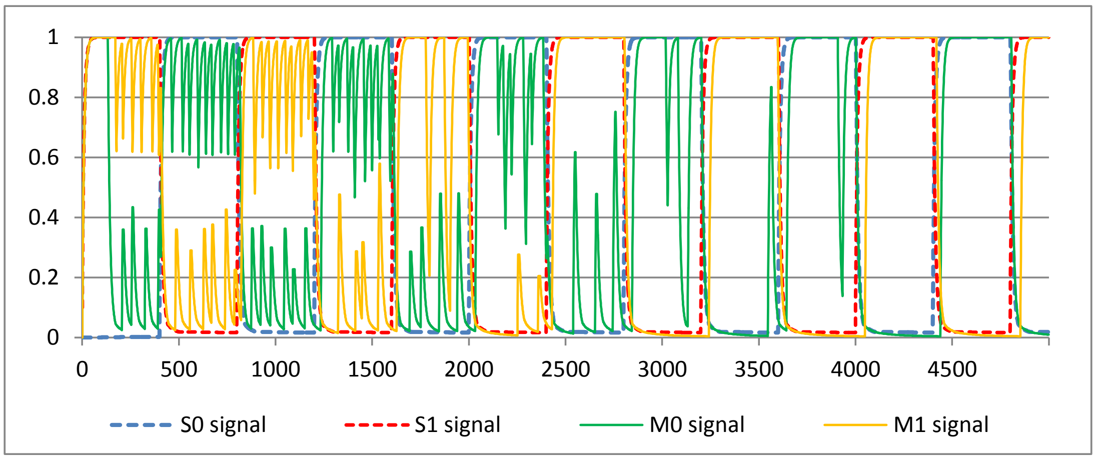
$$\mathbf{i^0} = (0,1)\,, \mathbf{i^1} = (1,0)\,, \mathbf{e^0} = (0.1,0.9)\,, \mathbf{e^1} = (0.9,0.1)\,.$$

[1]Recall the genome description format we described in the Section 4.1.1. The coordinates of the somata and axons are relative to the preceding soma coordinates, preceding axon coordinates when the preceding neuron is sensory, or to the point $[0,0]$ in the case of the first neuron.

Figure 5.1: *SimpleWorld in different stages of execution.* The first three images depict the conductive field of positive case of the simple experiment after a) the first step, b) seven hundred steps c) ten thousand steps. The image d) represents the negative case after ten thousand steps. The cells are represented by squares painted in grey scale, the lighter the cell is, the more potential is present. The conductance is depicted by small rectangles located on the edges of the grid. Conductance between two cells is represented by the color of the rectangle that lies in between them. The more the rectangle is light and yellow, the stronger the connection between the cells is. Somata and axons are painted in green and red respective. Active neurons are depicted in light colors while the inactive neurons are dark.



a)



b)

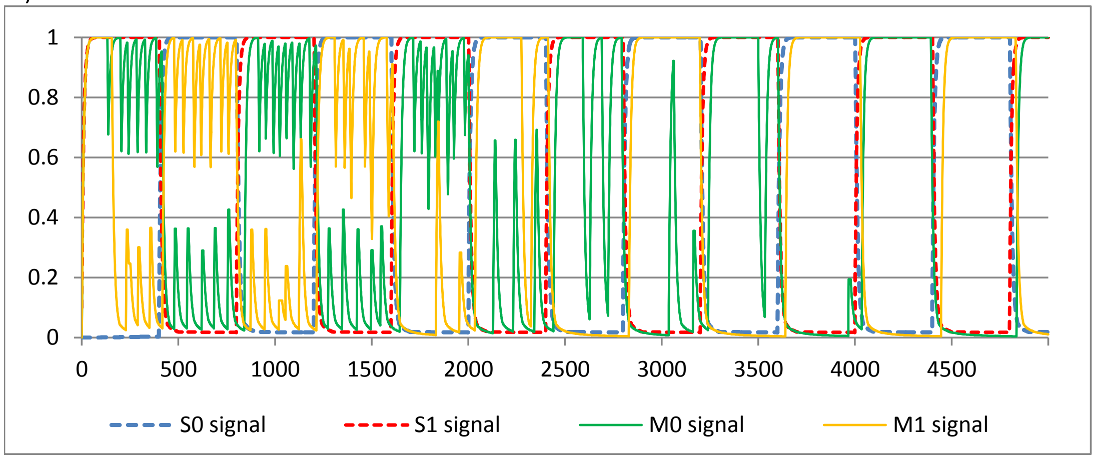


Figure 5.2: *Comparison of the sensory and motor activity in the SimpleWorld example.* The images a) and b) stand for positive case, c) and d) stand for the negative case. The y-axis describes the strength of the signal and the x-axis represents the steps in time. The blue and yellow lines show the strength of the sensory signal $S0$ and $S1$ respectively. The red line stands for the output of the motor 0 in the images a), c) and for the output of the motor 1 in the images b) and d).

Figure 5.3: *The somatic potentials in the SimpleWorld.* The vertical axis stands for the level of potential and the horizontal axis represents steps of the computation. The blue line represents the somatic potential of the motor neuron 0 and the yellow line represents the somatic potential of the motor 1. The data come from the first ten thousand steps of the positive case.

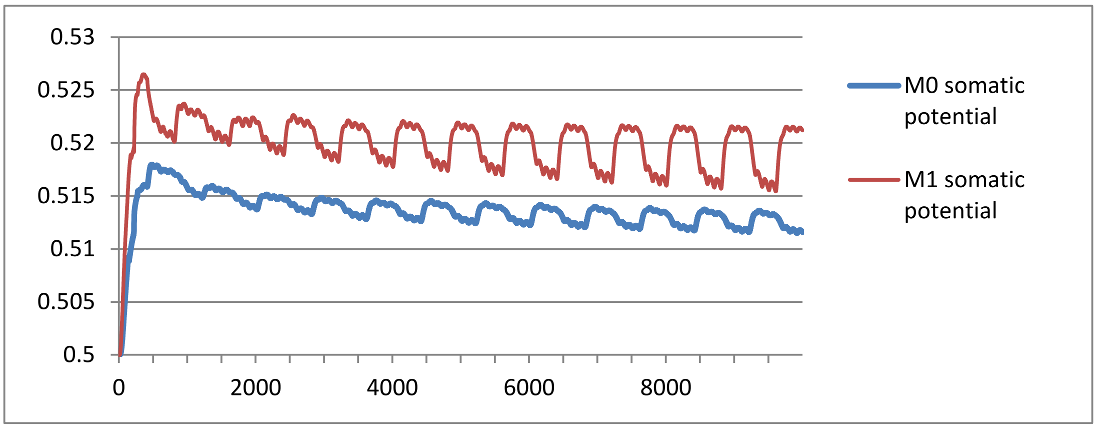
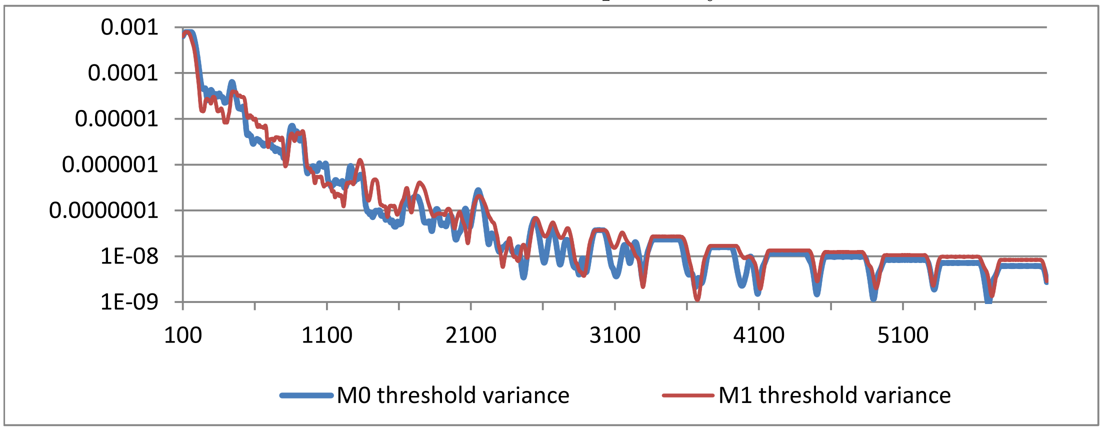


Figure 5.4: *The thresholds in the SimpleWorld.* The vertical axis stands for variance of the underlying quantity over last 100 steps. The horizontal axis represents steps of the computation. The blue and red lines represent variance of thresholds of the motor neurons 0 and 1 respectively.



We consider two cases of the experiment. A *positive* case, where we expect the model to assign the outputs to the inputs in such a way that they form two pairs $(\mathbf{i^0}, \mathbf{e^0})$ and $(\mathbf{i^1}, \mathbf{e^1})$; and a *negative* case, where we expect to obtain the pairs $(\mathbf{i^0}, \mathbf{e^1})$ and $(\mathbf{i^1}, \mathbf{e^0})$. The feedback f_{t+1} is computed using the last outputs of the model $\mathbf{u}$ and the expected outputs $\mathbf{e}$ as follows

$$f_{t+1} = -\sum_{j=1}^{s} \frac{|e_j - u_j|}{s}, \tag{5.1}$$

where s is the size of the output. In fact, the size of the output is always two in this example and the formula can be written in a simpler way. We introduce it in this general form because we reuse it in multiple examples and we want to point out that the feedback is computed in the same fashion in most of them.

The two pairs of input and expected output are exposed in alternating phases. A single phase is 400 steps long and we switch to the other phase immediately after the preceding phase ends. We run the adaptation for 50 such phases. Then we keep the thresholds and the conductance fixed and we continue with the computation feeding the same input. The states of the model in various phases of the computation are depicted in the Figure 5.1.

We can observe paths from the motor somata that receive the signal to the sensory axons which produce it. The path being built finds its expression in enlarging amplitude of the somatic potential of the motor neurons. The arithmetic mean of somatic potential of the motor neuron $M0$ during two consecutive phases in the beginning (phase 5 and 6) differs by $2 \cdot 10^{-4}$ and the difference between the phases 39 and 40 is more than 10^{-3}. The situation is illustrated in the Figure 5.3.

We use the Pearson's correlation coefficient to express dependence of the outputs on the inputs. The coefficient measures linear dependence between two quantities. The coefficient is 1 if the relationship is a perfect positive linear dependence and it is -1 if the linear dependence is negative. If the variables are independent, the coefficient is zero. The correlation coefficient between the signal strength of the sensory neuron $S0$ and the somatic potential of the motor neuron $M0$ is 0.785 in the evaluation part of the experiment (when the thresholds and the conductance are fixed). It is 0.804 between the other sensory-motor pair.

The values of the thresholds are determined at the same time as the conductance is adapted. They change a lot in the beginning and every time the change of a threshold is detected to be in different direction than it was the last step, the magnitude of the change is decreased. For that reason, the threshold level is stabilized during the computation. The resulting behavior is illustrated in the Figure 5.4. The lines in the graph depict variances of the thresholds in a sliding window that is a hundred steps wide. The variance drops from 10^{-3} to 10^{-9} during the first 5500 steps.

The activity of the two neurons adapts to follow the right neuron. The timings of the two motors in positive and negative case of this example are depicted in the Figure 5.2. The correlation coefficient between the signal strength of sensor $S0$ and motor $M0$ is 0.81 during the evaluation part and it is 0.722 between the other sensory-motor pair.

Because we get similar results for both positive and negative cases of the experiment and because the architecture of the genome is left-right symmetric, we can conclude that the adaptation method succeeded to adapt the model for the given problem. The correlation coefficients are summarized in the Table 5.1.

Positive case		
X	Y	corr(X,Y)
Sensor 0 signal strength	Motor 0 somatic potential	0.785
	Motor 0 signal strength	0.811
Sensor 1 signal strength	Motor 1 somatic potential	0.804
	Motor 1 signal strength	0.722

Negative case		
X	Y	corr(X,Y)
Sensor 0 signal strength	Motor 1 somatic potential	0.800
	Motor 1 signal strength	0.820
Sensor 1 signal strength	Motor 0 somatic potential	0.827
	Motor 0 signal strength	0.731

Table 5.1: Correlation coefficients in the Simple experiment.

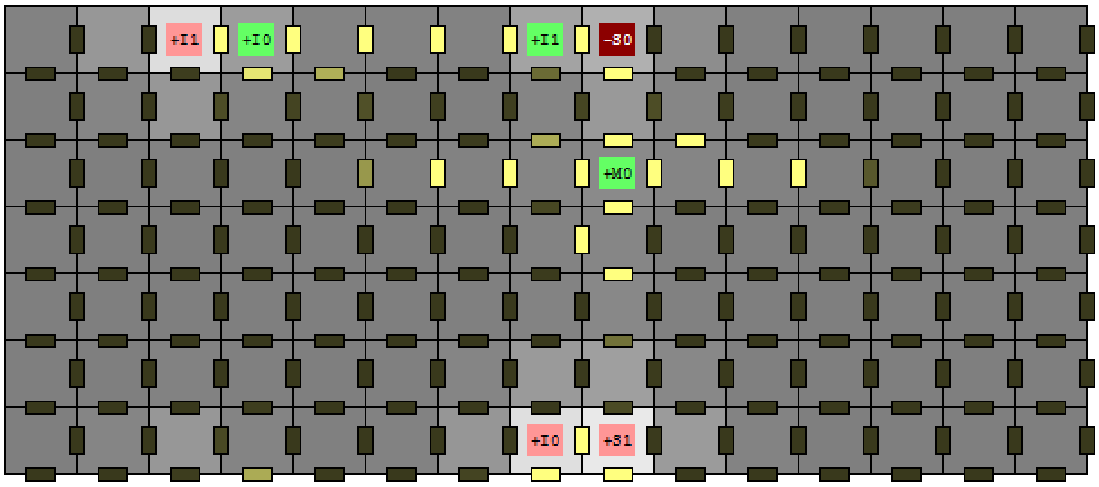
5.2 SwitchWorld

The *SwitchWorld* experiment demonstrates the ability of the conductive field brain to preserve an information over time. In particular, we look for a circuit which exhibits an ability to receive *switch-on* and *switch-off* signals. It generates a signal by the time it is turned on and it stays calm when it is turned off.

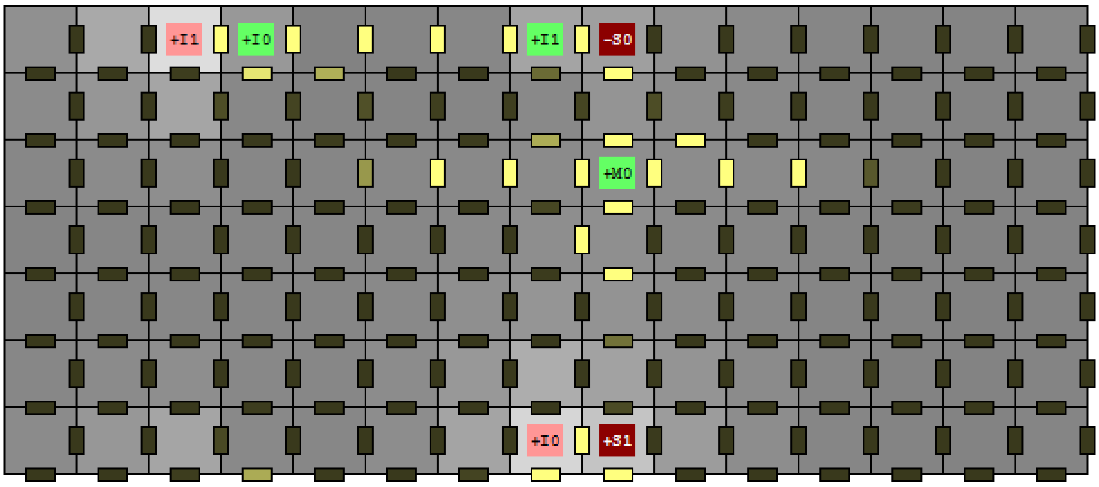
There are two sensory inputs to the model. One of them receives the switch-on signal while the other accepts the switch-off signal. We define three input vectors reflecting that situation, the *switch-on input* $\mathbf{i_1} = (0,1)$, the *switch-off input* $\mathbf{i_2} = (1,0)$, and the *continue input* $\mathbf{i_0} = (0,0)$. There is a single output motor which is either expected to stay calm $\mathbf{e_0} = (0)$ or to produce signal $\mathbf{e_1} = (1)$. For illustration, see the resulting grid in the Figure 5.5. Length of the signal phase is 150 steps and the length of the continue phase is 350 steps. The phases do not alternate regularly to ensure that the brain does not learn the period and it really reacts to the input signals. Instead, the polarity of each phase is generated randomly and the switch-on or switch-off signals are produced only between two phases of different polarity.

We employ the Equation 5.1 from the SimpleWorld experiment to define the objective function. Since there is only a single motor output in this experiment, it reduces to $-|e_j - u_j|$. The evolution algorithm without crossover is used to find an appropriate genome. After four generations with 15 individuals, three elite genomes and fixed conductive field size to $[15, 7]$, we arrive at the following genome.

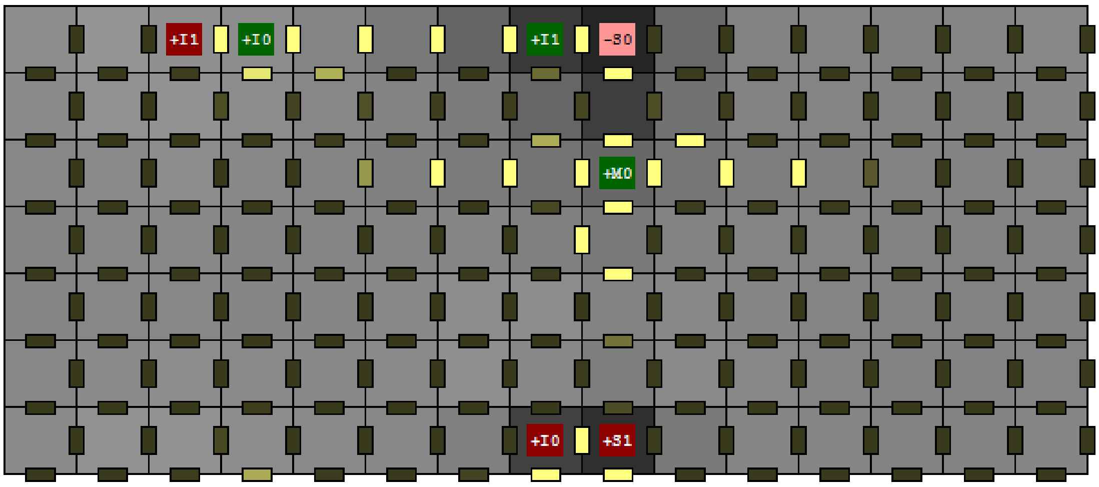
```
Genome(S:2 M:1 [15, 7]):
  +Sensory(Axon[8, 6] SensorIndex:1 Polar: Excit)
  +Sensory(Axon[15, 1] SensorIndex:0 Polar: Inhib)
  +Internal(Soma[10, 0] Axon[4, 6] Polar: Excit)
```



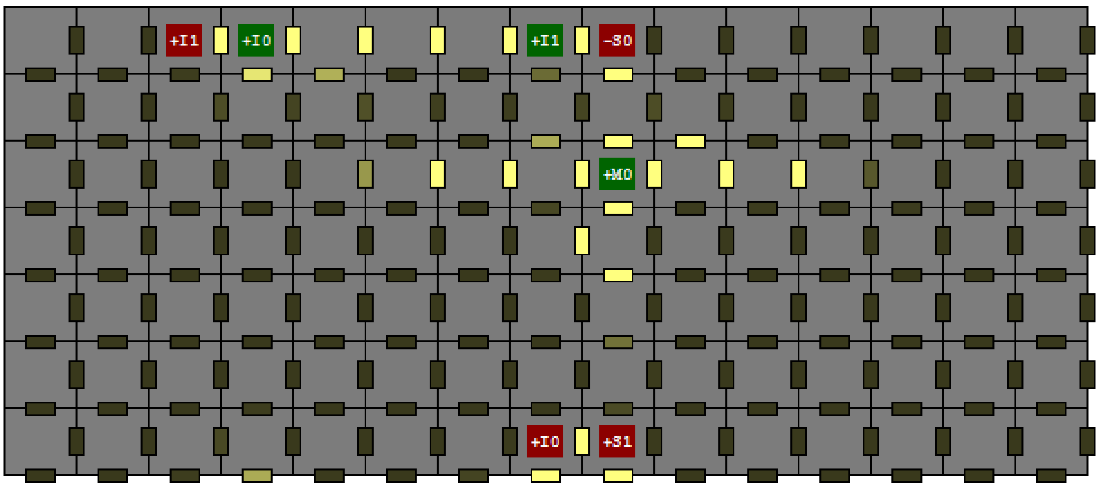
(a)



(b)



(c)



(d)

Figure 5.5: *SwitchWorld in different stages of execution.* The conductive field is depicted when a) switch-on signal is received, b) continues in the active phase, c) switch-off signal is received, d) continues in the passive phase.
The cells are represented by squares painted in grey scale, the lighter the cell is, the more potential is present. The conductance is depicted by small rectangles located on the edges of the grid. Conductance between two cells is represented by the color of the rectangle that lies in between them. The more the rectangle is light and yellow, the stronger the connection between the cells is. Somata and axons are painted in green and red respective. Active neurons are depicted in light colors while the inactive neurons are dark. The labels containing S, I, and M stand for sensory, internal, and motor neurons respectively. The signs $+$ and $-$ represent excitatory or inhibitory neuron.

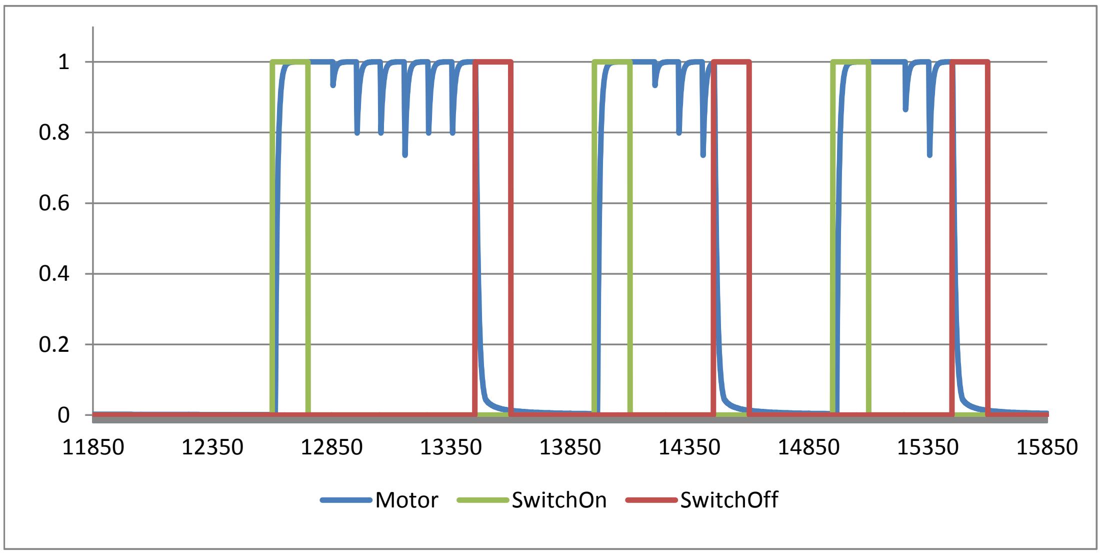


Figure 5.6: *Comparison of the sensory and motor activity in the SwitchWorld example.* The y-axis describes the strength of the signal and the x-axis represents the steps in time. The red and green lines show the strength of the sensory signal $S0$ and $S1$ respectively. The blue line stands for the output of the motor signal $M0$.

```
+Internal(Soma[4, 0] Axon[10, 7] Polar: Excit)
+Motor(Soma[1, 2] MotorIndex: 0 Polar: Excit)
```

The resulting brain is depicted in the Figure 5.5 in four stages of computation, that is, when receiving each of the two switch signals, continuing in the activity, or staying calm. Notice that the sensory neuron connected to the switch-off input is inhibitive. The internal neurons seem to form a loop that is fired by the switch-on signal and interrupted by the switch-off signal. The correlation between the expected output and the actual output during the evaluation is 0.958.

We illustrate the output signal in the Figure 5.6. The green line represents the switch-on signal, the red line stands for the switch-off signal, and the blue line depicts the motor output. The motor output quickly raises during the switch-on signal and it rapidly drops during the switch-off signal. In the periods when no input signal is received, the motor output continues in the same level of activity. This behavior is exactly what we described in the beginning of the section.

5.3 BinBoolWorld

In the *BinBoolWorld* experiment, we demonstrate the ability of the conductive field brain to solve elementary logical tasks in practice. Let us have $j \in \{1, \ldots, 16\}$ and let $\mathbf{b_j} = (b_{j,0}, b_{j,1}, b_{j,2}, b_{j,3})$ be a vector corresponding to a binary representation of j, where the most significant bit is $b_{j,3}$. Then we describe a *binary boolean function with index* j using the Table 5.2. We assign a motor output $M0$ to each pair of sensory inputs $S0$ and $S1$.

Assume we have an input vector $\mathbf{i} = (i_1, i_2)$, then we define an augmented input vector $\mathbf{a} = (a_1, a_2, 1)$. The reason why we feed the conductive field brain with augmented vector is that it supplies a steady signal for the situation when

$S0$	$S1$	$M0$
0	0	b_1
0	1	b_2
1	0	b_3
1	1	b_4

Table 5.2: Binary boolean function with index j.

$b_1 = 1$. There is expected an output signal but no input would be fed. As we have seen in the previous case, it is possible to generate a signal without a steady input. We therefore don't add any external functionality but we simplify the situation.

A phase when a pair of input and corresponding expected output are exposed is 350 steps long. First, an adaptation period is executed. It consists of 24 phases, 6 for each input and expected output pair. The order of the adaptation phases is random. Then we run an evaluation period. It is composed of twelve phases, three for each input and expected output pair. If the evaluation passes a satisfactory condition, then the examined genome is returned. Otherwise, an other cycle of adaptation and evaluation is repeated. We run at most four such cycles. The evaluation computes the average output of phases when 1 is expected and and when 0 is expected. Then it takes minimum from the 1-phases and maximum form the 0-phases and return a difference between them. If there is no phase with expected output 0, then the minimum of 1-phases is returned and vice versa. To fulfill the condition on the fitness values, the result is divided by two and 0.5 is subtracted, as illustrated by the following formula

$$\frac{min\,\{\text{avg outputs of 1-phases}\} - max\,\{\text{avg outputs of 0-phases}\}}{2} - 1.$$

The Table 5.3 summarizes the results of the experiment. It shows the correlation coefficients between the outputs of the adapted conductive field brains acquired during evaluation. The indexes 0 and 15 are left out, because their output is the same every step. For that reason, the variance becomes zero and the correlation coefficient is not defined. The fact, that the highest number of every row lies on the diagonal (the same index of row and column), means that the model is adapted to the corresponding function the most, in terms of linear dependency.

The indexes 6 and 9 turned up to be the most problematic. Their correlation coefficients are 0.834 and 0.884 resp. and it took much longer to obtain their genomes. While the arithmetic mean of the execution time of the algorithm of the rest indexes was less than 33 seconds, it took approximately 45 minutes to find each of the genomes for the indexes 6 and 9. The table 5.4 summarizes the execution times of the evolutionary algorithm. The indexes 6 and 9 stand for the functions representing the logical XOR and identity. It is known, that these functions are not linealy separable [19]. Moreover, it is proved that they are not computable by a single perceptron, which suggests that there is a need for multiple neurons in the conductive field brain. The resulting genomes correspond to that statement. The indexes 6 and 9 were the only two that led to genomes with internal neurons. The genomes we obtained in this experiment can be found on the CD attached to this text.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0.974	−0.312	0.573	−0.32	0.567	−0.547	0.342	−0.342	0.547	−0.567	0.32	−0.573	0.32	−0.974
2	−0.32	0.978	0.57	−0.343	−0.574	0.55	0.315	−0.315	−0.55	0.574	0.343	−0.57	0.343	0.32
3	0.556	0.57	0.975	−0.572	−0.014	−0.002	0.554	−0.554	0.002	0.014	0.572	−0.975	0.572	−0.556
4	−0.301	−0.346	−0.56	0.968	0.578	0.539	0.321	−0.321	−0.539	−0.578	−0.968	0.56	−0.968	0.301
5	0.581	−0.55	0.026	0.539	0.969	−0.01	0.569	−0.569	0.01	−0.969	−0.539	−0.026	−0.539	−0.581
6	−0.395	0.456	0.053	0.508	0.098	0.834	0.569	−0.569	−0.834	−0.098	−0.508	−0.053	−0.508	0.395
7	0.348	0.318	0.577	0.289	0.552	0.525	0.955	−0.955	−0.525	−0.552	−0.289	−0.348	−0.289	−0.348
8	−0.294	−0.34	−0.548	−0.34	−0.548	−0.588	−0.973	0.973	0.588	0.548	0.34	0.548	0.34	0.294
9	0.444	−0.393	0.044	−0.627	−0.159	−0.884	−0.577	0.577	0.884	0.159	0.627	−0.044	0.627	−0.444
10	−0.559	0.54	−0.017	−0.544	−0.955	−0.003	−0.563	0.563	0.003	0.955	0.544	0.017	0.544	0.559
11	0.321	0.323	0.558	−0.967	−0.559	−0.558	−0.323	0.323	0.558	0.559	0.967	−0.558	0.967	−0.321
12	−0.562	−0.561	−0.973	0.568	0.005	0.006	−0.555	0.555	−0.006	−0.005	−0.568	0.973	−0.568	0.562
13	0.336	0.308	0.557	−0.967	−0.547	−0.571	−0.324	0.324	0.571	0.547	0.967	−0.557	0.967	−0.336
14	−0.975	0.325	−0.563	0.339	−0.551	0.575	−0.311	0.311	−0.575	0.551	−0.339	0.563	−0.339	0.975

Table 5.3: The table summarizes correlation coefficients between the ideal outputs (columns) and the real outputs of conductive field brains evolved and adapted for a particular binary boolean function (rows). The first row and the first column represent the index of the function.

Index	Time in sec	Index	Time in sec
0	1.018	8	2.715
1	19.250	9	2,656.799
2	10.429	10	54.216
3	56.982	11	7.953
4	23.328	12	67.696
5	102.000	13	4.680
6	2,705.699	14	22.859
7	84.857	15	1.314

Table 5.4: Execution time of the evolutionary algorithm. The first column represents the index of a binary boolean function, the second column contains the time interval in seconds.

5.4 UniBoolWorld

The experiment *UniBoolWorld* extends the *BinBoolWorld*. We search for an *universal* genome which can adapt to multiple indexes of the binary boolean function. We supply two indexes, 8 and 14, which stand for AND and OR logical operators respectively. The objective function is the same as in the *BoolWorld* experiment except it runs separately for every index an then it returns the minimum.

We obtain a genome

```
Genome(S:3 M:1 [3, 5]):
 +Sensory(Axon[1, 1] SensorIndex:0 Polar: Excit)
 +Motor(Soma[1, 0] MotorIndex: 0 Polar: Excit)
 +Sensory(Axon[0, 1] SensorIndex:1 Polar: Excit)
 +Sensory(Axon[1, 3] SensorIndex:2 Polar: Excit)
```

The correlation coefficients between the outputs obtained during evaluation and the desired outputs are 0.924 and 0.897 in the case of OR and AND respectively.

Conclusion

In this thesis, we have developed and examined properties of a model of brain tissue containing a spatial information. The motivation for creation of such a model is the hypothesis that the spatial properties of the brain tissue may play an important role in the learning and memorizing processes. We focus on the dendrite shape as well as its modification and we seek for a strategy that may eventually help to explain how the human brain learns and memorizes.

We have searched for what is known in the area of brain modelling and is related to our idea. We have found two famous and deeply studied groups of models, the neural networks and the cellular automata. The conductive field brain model, which we developed, combines ideas from both. Each other enriches so much that we are not able to assign the conductive field brain to any of the groups.

To be able to study and evaluate our idea by means of computer science, we define the model formally. We observe, that there exists a single time step function which gives the next configuration and an output for any correct combination of current configuration and input. Later we prove a theorem which states that the conductive field brain placed to an appropriate environment is as strong as the Turing machine.

Having explored the theoretical capabilities of the model, we propose algorithms for searching the right configuration for a given problem. We determine size of the model and locations of neurons using evolutionary algorithms and we adapt the shape of dendrites and their sensitivity separately. We implement the model including the evolution and adaptation algorithms to demonstrate its properties in a simulated environment. The experiments show that the conductive field brain is capable of adaptation of a single architecture to multiple problems based on the inputs and the feedback from the environment. The simulations also demonstrate that the evolutionary algorithms can deliver architectures, which are then adaptable to solve various tasks.

Some limitations of this work can be found in the evolutionary search for the right architecture. Firstly, we experiment with elementary problems and the structural development of a brain is left back. However, we propose several future steps in this area along with the description of implemented algorithms. A notion of modules as analogy to Holland's schemes in the genetic algorithms is presented. Patterns representing populations of neurons which together form a structure for solving a complex task may emerge in the genome. That idea brings a lot of questions to be answered.

Secondly, we have shown some theoretical capabilities of the model and we proposed an algorithm for its adaptation. We recommend a future theoretical study of the adaptation algorithm. What are the limits of the adaptation? How much information has to be coded in the genome and what can be acquired from the environment? We could try to find a capacity measure of a conductive field brain. Such a measure would express the amount of information that the model could store.

Next, we work with the model in a fully synchronised manner. Every step the whole conductive field is recomputed and all the neurons test the somatic potential level. A question may be posed whether it can't be done in an asynchronous way. The Hopfield networks may serve as inspiration. Every step, a single neuron updates its state based on the neighborhood. Moreover, the computation could be executed in parallel. We have worked with configurations in scope of only a few neurons. The parallel and asynchronous approaches may enable us to study behavior of the model in larger scale.

Various models inspired by the biological concept of many interconnected computing units exist. Some of them are well studied from the computer science point of view and they often serve as mathematical tools for solving complex problems. The source of ideas is biology and the target is computing. We aim to make a step in closing a loop back to biology. We inspire in the organic brain, then we create a simplified model where we study the principles of how it works. We develop an algorithm that solves an issue and we close the loop by proposing the principle of the algorithm back to biology to be examined.

One of the key principles we propose is the algorithm for determining the shape of dendrites. It is the core of adaptation of our brain model to environmental conditions. This opens a question whether an analogous process couldn't be found in nature. The idea behind is that an increase in concentration of a chemical signalling a positive feedback from the environment in combination with change of activity of a neuron may lead to production of a new substance in the neural cell. This substance would be than consumed by those parts of the neural membrane, which received the signal to change the activity. This way, the open problem of precise feedback delivery in the brain, could be addressed. Such a hypothesis needs an extensive interdisciplinary discussion to be accepted or proved wrong.

Bibliography

[1] Alonso-Sanz, Ramon. *Discrete systems with memory [online].* Singapore: World Scientific, c2011, xi, 465 pages.

[2] Bear, Mark F., Barry W. Connors, Michael A. Paradiso. *Neuroscience: exploring the brain.* 3rd ed. Philadelphia: Lippincott Williams & Wilkins, 2007, xxxviii, 857 s. ISBN 9780781760034.

[3] Beigy, Hamid, M. R. Meybodi. *A Mathematical framework for cellular learning automata.* Advances in Complex Systems, Vol. 7, Nos. 384, 2004, Pages 295–319.

[4] Chua, Leon O., Lin Yang. *Cellular Neural Netwotks: Theory.* IEEE Transactions on Circuits and Systems, vol.35, no.10, October 1988, Pages 1257-1272.

[5] Chua, Leon O., Lin Yang. *Cellular Neural Netwotks: Applications.* IEEE Transactions on Circuits and Systems, vol.35, no.10, October 1988, Pages 1273-1290.

[6] Cimagalli, Valerio, Marco Balsi. *Cellular Neural Netwotks: A Review.* Proceedings of Sixth Italian Workshop on Parallel Architectures and Neural Networks. Vietri sul Mare, Italy, May 12-14, 1993, World Scientific

[7] Di Gregorio, Salvatore, Roberto Serra *An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata.* Future Generation Computer Systems 16 (1999) 259–271

[8] Eiben, Agoston E., J. E. Smith. *Introduction to Evolutionary Computing* Springer Berlin Heidelberg, 2003, ISBN:3540401849.

[9] FitzGerald, Thomas H.B., Karl J. Friston, Raymond J. Dolan. *Characterising reward outcome signals in sensory cortex.* NeuroImage, Volume 83, December 2013, Pages 329–334.

[10] Friston, K. J., G. Tononi, G.N. Reeke, O. Sporns, G.M. Edelman. *Value-dependent selection in the brain: simulation in a synthetic neural model.* Neuroscience, 59 (1994), Pages 229–243

[11] Hopcroft, John E, Rajeev Motwani a Jeffrey D. Ullman. *Introduction to automata theory, languages and computation.* 2nd ed. Boston: Addison-Wesley, c2001, xiv, 521 pages. ISBN 0-201-44124-1.

[12] Knott, Graham, Anthony Holtmaat. *Dendritic spine plasticity - Current understanding from in vivo studies.* Brain Research Reviews, Volume 58, Issue 2, August 2008, Pages 282–289.

[13] McAllister, Kimberly A. *Cellular and Molecular Mechanisms of Dendrite Growth.* Cerebral Cortex, 10:963-973, 1047-3211/00, October 2000, Oxford University Press.

[14] McCulloch, Warren, Walter Pitts *A Logical Calculus of Ideas Immanent in Nervous Activity.* Bulletin of Mathematical Biophysics 5 (4), Pages 115–133, 1943.

[15] Mitchell, Melanie. *An introduction to genetic algorithms.* Cambridge: MIT Press, 1998, viii, 209 pages, ISBN 0262631857.

[16] Parsons, Jay A., Mark A. Fonstad *A cellular automata model of surface water flow.* Hydrological Process 21, John Wiley & Sons, Ltd., 2007, Pages 2189–2195.

[17] Popov, Igor. emphThe Problem of constraints on variation, from Darwin to th present. Ludus Vitalis, vol. XVII, num. 32, 2009, pp. 201-220.

[18] Roelfsema, Pieter R. , Arjen van Ooyen, Takeo Watanabe. *Perceptual learning rules based on reinforcers and attention.* Trends in Cognitive Sciences, Volume 14, Issue 2, February 2010, Pages 64-71.

[19] Rojas, Raúl. *Neural networks: a systematic introduction.* Berlin: Springer, 1996, xx, 502 pages, ISBN 3540605053.

[20] Rolls, Edmund T., Alessandro Treves *The neuronal encoding of information in the brain.* Progress in Neurobiology, Volume 95, Issue 3, November 2011, Pages 448-490.

[21] Russel, Stuart and Peter Norvig. *Artificial intelligence: a modern approach. 2nd ed.* Upper Saddle River: Prentice-Hall, 2003, xxviii, 1081 s. ISBN 0-13-790395-2.

[22] Siegelmann, Hava T. *Neural networks and analog computation:beyond the Turing limit* Boston: Birkhäuser, 1999, xiv, 181 pages. ISBN 0817639497.

[23] Sipser, Michael. *Introduction to the theory of computation.* Boston: PWS Publishing Company, 1997, xv, 396 pages. ISBN 053494728x.

[24] Storn, Rainer, Kenneth Price. *Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces.* Journal of Global Optimization, Volume 11, Issue 4, 1997, Pages 341-359.

[25] Tomás, Pedro, Leonel Sousa. *A quantitative analysis of firing rate estimators: Unveiling bias sources.* Neurocomputing, Volume 73, Issues 16–18, October 2010, Pages 2944-2954.

[26] Wooldridge, Michael. *An introduction to multiagent systems.* Chichester: Wiley, 2002, xviii, 348 Pages. ISBN 0-471-49691-x.

[27] Yang, X. S., Y. Z. L. Yang. *Cellular automata networks* Proceedings of Unconventional Computing, Luniver Press, 2007, Pages 280-302.

[28] Yasunaga, Kei-ichiro, Takahiro Kanamori, Rei Morikawa, Emiko Suzuki, Emoto Kazuo. *Dendrite Reshaping of Adult Drosophila Sensory Neurons Requires Matrix Metalloproteinase-Mediated Modification of the Basement Membranes.* Developmental Cell 18, April 20, 2010, Elsevier Inc., Pages 621–632.

Symbols and notation

The symbols and notation are used in the following meaning throughout the thesis.

Term	**Meaning**
$\mathbb{N}$	natural numbers $\mathbb{N} := \{1, 2, 3, \ldots\}$
$\mathbb{R}$	real numbers
$\mathbf{v}$, s, M	vector, scalar, matrix resp.
$\langle a, b \rangle$	closed interval
(a, b)	open interval
$\mathcal{P}(S)$	power set of S
$(f \circ g)(x) = f(g(x))$	composition of functions

Appendices

A. Custom world

As advertised in the previous parts, the application was created with an intention of further development. Hereby we provide a brief introduction in coding of a custom experiment. We start with an empty shell of a class that inherits `World`

```
using System;
using System.IO;
using CoFiBa.Environment;
using CoFiBa.Genome;
using CoFiBa.Brain;
using CoFiBa.Utils;

namespace CoFiBa_App.Worlds
{
  class MyWorld:World
  {
    public override double SatisfactionCondition
    { get { return 0; } }

    public SimpleWorld(TextWriter log, ConfigurationAdapter config)
      : base(log)
    { }

    public override Genome NewGenome()
    {  }

    public override double ObjectiveFunction(Genome genome,
      Controller controller = null)
    { }
  }
}
```

The first member of the class, the property `SatisfactionCondition`, contains only the get clause. The value is used to stop the evolution when a genome with grater or equal fitness is found. We skip the constructor for a while and consider the `NewGenome` method. It serves to generate a new genome. The method is typically called when the first population is being initialized. Since it is called repeatedly, it is recommended to randomize the new genome. The genome constructor can do the job when with appropriate parameters.

The constructor serves to initialize the common variables like some sample data. It is also place where the world configuration is read from the `config` parameter. The `Log` parameter is recommended to be stored for later use. It prints to the console and a log file if it is opened.

The main purpose of the World classes is to provide the `ObjectiveFunction`. The function takes a genome and a controller and returns fitness of the genome. We recommend two best practices. The first is to initialize a local randomizer by `genome.AdaptSeed` to generate the learning data set. The seed will be eventually printed into log so the experiment can be repeated. Randomizer initialization follows.

```
// init local randomizer
localRandom = new Random(genome.AdaptSeed);
```

The next best practice is to cultivate the brain in the `InSilicoLab` initialized with the controller received in parameter. This enables the GUI analyzer to communicate with the lab and control the execution remotely.

```
if (controller == null)
  controller = new Controller((AnalysisSeries)null);
InSilicoLab lab = new InSilicoLab(genome, controller);
```

When the new `MyWorld` class is finished, it needs to be registered in the program. To do that, add a case

```
 case "myworld":
          world = new MyWorld(logger, config);
          break;
```

to the `loadWorld` method of the class `Program`.

B. Manual

The program is run from a windows command-line with two arguments

- a log file with the flag `-l` and a file path of the desired file,
- a configuration file with the flag `-c` and a file path of the desired configuration file.

Both the files are text files. A sample command would be

```
cofiba_app -l "C:\MyLogFile.log" -c "C:\MyConfigFile.config"
```

The configuration file determines what environment is launched and what world is simulated. Then there are custom configurations for the environment and the world specified earlier. Each configuration is on a single line. A line can contain a configuration, white space, or a comment. Comment line begins with two slashes (`//`). A configuration line consists of a configuration name, equal sign, and a value. The names are not case sensitive, but the values are. A sample configuration file with comments is present on the attached CD in the folder `Configurations`.

B.1 Analysis GUI

The environment for analysis contains a graphic user interface. The interface is launched right after the application loads the environment. It consists of a panel where the model simulation is animated. A snapshot of that animation is described in the Figure 5.1. Such a snapshot is created when clicked to the first menu button `Snapshot`. The animation starts when the `GoSteps` button is clicked. It does a given number of steps and waits for a specified number of milliseconds before each step to slow down the animation. The waiting interval can be set to zero.

The second menu button `Analyze series` opens a series analysis dialog. There is a list of available series. The names obey the following rules

- `Sen` and `Mot` stand for sensors and motors signal strength,
- `S`, `I`, `M` represent sensory, internal, and motor neurons, where
  - `spot` is somatic potential,
  - `axpot` is axonal potential,
  - `sig` is signal strength,
  - `thr` is threshold value.

The `Repaint` button needs to be clicked after every change of selection. Then a chart is painted. The chart area can be zoomed to a specified rectangle and the resulting image can be exported to a *PNG* file when the menu button `Export` is clicked and the option `Graph` is chosen. When we select the `Data` option in the export menu, we can export the selected series data to a *CSV* file.

C. Contents of the attached CD

The contents of the attached CD are:

- Text of the thesis in the file `PavelJohn_MasterThesis.pdf`.
- CoFiBa application installation (recommended) is in the folder `Application_Install`.
- CoFiBa application executables are located in the folder `Application_ExecutableOnly`. Execution does not need any installation, but the compatibility with various .NET Framework versions may be problematic.
- Source code of the application is contained in the folder `CoFiBa`.
- The configuration files for each of the four experiments are stored in the `Configs` folder.
- Genomes obtained in the BinBoolExperiment are located in the file `BinBoolWorld_Genomes.txt`. The file includes randomized seed for the possibility of exact repeat of the experiments.