EVOLUTIONARY NEURAL NETWORKS
1. INTRODUCTION v.2

José del Carmen Rodríguez Santamaría
https://www.ejristos.com
Bogotá, Colombia

November 25, 2024

Abstract

We work out various models of a neuron and its networks. We show in the first instance a mechanism that can recognize a pattern. It is a Boolean Circuit composed of boolean gates. Then, we provide boolean gates with stability against noise and we get a perceptron. Next, we propose a mechanism that can recognize a pattern even in the presence of noise and variability. It is a neural network (NN). To end, we show how to tie Evolution to an NN to produce a mechanism that learns to recognize a pattern amid noise and variability. It is an Evolutionary Environment for Neural Networks (EENN). Everything is shown over a toy model. This introduction is targeted to persons that want to understand both the whys and the hows of Evolutionary NNs. In the first version of this study we could not get Evolution to solve an important problem, but for the second version, we did it, see 64. A wrong but strategic sign was corrected for the third version.

Contents

1 BOOLEAN PATTERN RECOGNITION 4
- 1.1 Our task . . . 6
- 1.2 The architecture of the boolean classifier . . . 7
- 1.3 The Boolean Classifier . . . 7

2 NOISE AND VARIABILITY 11
- 2.1 Solution with boolean circuits . . . 12
- 2.2 Solution with Turing Machines . . . 13
- 2.3 Abstracting the neuron . . . 15
- 2.4 The perceptron . . . 16
- 2.5 The perceptron generalizes boolean gates and neurons. 21
- 2.6 Neural Networks (NN) . . . 23
- 2.7 Building simplicity . . . 24
- 2.8 Perceptron as a JS class . . . 28
- 2.9 The code for a NN as JS class . . . 35

3 LEARNING 36
- 3.1 Back Propagation Algorithm . . . 37
- 3.2 Evolutionary Environments for Neural Networks (EENN) 38
  - 3.2.1 Implementation . . . 39

4 CONCLUSION 46

5 ANSWERS TO EXERCISES 47

6 THE BIBLIOGRAPHY 58

INTRODUCTION

Recognizing patterns is at the root of our human culture. Once I asked a little girl to share the torte for her birthday with me but she answered: `I don´t know you`. This means that, unexpectedly for me, I did not figure in her records of patterns of persons. As a consequence, I was not admitted to her party.

Our general purpose is to automate pattern recognition. Our departing idea is to use the language but be restricted to boolean expressions, which are those that refer just to be or not to be. Example: about a scientific article, whether is or not interesting, or whether it is or not clear. Thus, pattern recognition becomes a matter of propositional logic that can readily be automated thanks to boolean gates, those that deal with 1 and 0, TRUE and FALSE. This is illustrated in an example in *Section 1: Boolean Pattern Recognition*.

This type of recognition has not too much to do with what we are: our little girl would invite her mom to her party despite the brand-new dress that she is wearing today. So, she can recognize her mother regardless of noise. Thus, we will show that boolean recognizing can be successfully modified thanks to the study of biological neurons. They can recognize patterns even under noise and variability. This is possible thanks to an abstraction or a mathematical model of neurons that was called the **perceptron** and that generalizes boolean gates but are moreover stable in the presence of noise and variability. Circuits of perceptrons form a Neural Network (NN) and we will show in *Section 2: Noise and variability*, how to use them for pattern recognition.

A neural network calculates a function, an assignment from one set to another.

1 Example. *To recognize images in a data set, the function generated by the network associates a name to a given subset*

of images, say, the NN associates the string lion *to an image of a lion.*

The function that defines the aim of a NN is called the **spec or specification function**.

A further challenge is that we all learn even at very old ages. This means that from being very bad classifiers, we can receive training that improves our recognizing capabilities. Thus, *materialism prompted engineers to pursue a mechanism capable of achieving learning*. This was solved by the **Back Propagation Algorithm**, that gradually modified NNs to implement their spec function as desired by the designer. We do not implement that Algorithm. Rather, we show, in *Section 3: Learning*, how to use Evolution to modify a population of NNs by cloning the fittest, the best for solving the given recognizing task, to form the population of the next generation. In that way, the population is expected to improve classifying capabilities when generations run. Over a very simple architecture, a toy model, we test this expectancy.

1 BOOLEAN PATTERN RECOGNITION

Neurons were at first an idea formulated in 1740 by Emmanuel Swedenborg (Fodstad [3] 2001).

Purkinje gave in 1837 the first description of neurons, (Purkyně [10]).

More than 200 years of ensuing hard work in the study of electrophysiological properties of nerve cells can be succinctly summarized: `neurons are electrochemical machines`. They receive nutrients and oxygen and transform them into membrane potential. This is destabilized when the neuron is excited, an electrical spike appears that is next transmitted as a signal, biochemical most often, to other cells (Piccolino, [9] 1998).

This is materialism in full and bright colors although the details are still under discussion (Ghosh et al, [4] 2022).

To convert a belief into Science, the Scientific Method demands to make predictions. The most immediate and necessary one regarding neurons is that if a machine is made to realize the abstracted properties of neurons, the machine must think. Warren McCullock and Walter Pitts accepted in 1943 this challenge and proposed the McCullock-Pitts neuron. We will see in this section how to resolve a pattern recognition task with this model. Some definitions are in order:

A Pattern Recognition Task is solved by a spec function that assigns a name, a number, or a label to a given subset of images, sounds, or texts, or instances of a complex pattern. Say, to an image of a handwritten number that was drawn by an adult, the function assigns its name, say, five, or, eventually, its Arabic representation. This spec function must be implemented into a mechanism.

A Classifier is a mechanism that solves a Pattern Recognition Task.

A boolean value can be only 1 else 0, TRUE else FALSE, to be ON else OFF.

A bitmap is a sequence of Boolean values or bits that encode for an image.

2 Example. *A digital image whose pixels are in black else white generates a bitmap because all pixels put in a row generate a binary sequence.*

A Boolean Classifier is a classifier that accepts a bitmap as input and its output is given by a raw of numbered lamps that light if and only if the corresponding class has been detected.

3 Example. *In the task of recognizing handwritten numbers, we have an image with pixels in black or white. Each pixel generates*

a boolean value and the whole image is reported as a bitmap, a sequence or array of boolean values or bits. It is the input. As output, we have lamps in a row that are numbered 0, 1, ... 9, such that when the system detects, say, a 5, the corresponding lamp fires. Each lamp can be ON else OFF so, it represents a boolean output. We will develop an example that recognizes 1 or 0. This is our toy model all along this article. In consequence, we have two lamps in a row, the first fires when the system detects a 1 but remains turned off otherwise. The second lamp fires if and only if the system detects a 0. The input to the function is a bitmap, the output of each perceptron is a boolean value and the output of a NN is a binary sequence with the binary values of all perceptrons.

1.1 Our task

Let us define that *our task is to recognize bitmaps of images of numbers that represent 0 else 1.* Each pixel of each image can be depicted in white or black and the corresponding bitmaps are sequences of zeros or ones. A pixel that is colored in black is encoded as one, but if the pixel is blank, it is encoded as a zero. In this article, we will discriminate between two numbers only, 1 and 0. Each bitmap is checked against the two archetypes or images of 1 and 0 and the system will answer: the bitmap is 1, or the bitmap is 0, or nothing that corresponds to something else that is UNKNOWN. Our images will be drawn in a 3×3-binary grid, each pixel in black else white. See Figure 1.

4 Exercise. *Draw a zero in the aforementioned grid and find the corresponding binary encoding.* Answer

5 Definition. *The task of recognizing arbitrary binary patterns over a 3×3 grid that encodes a zero else a one is called here the* **0-else-1-problem**.

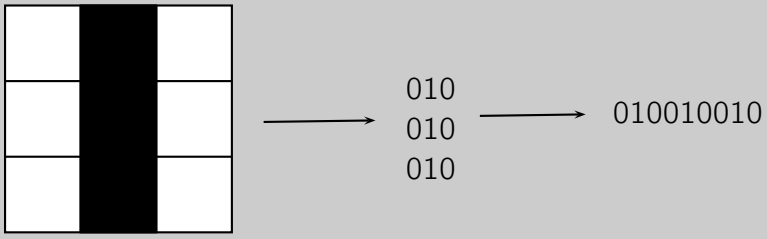


Figure 1: Number 1 is drawn in a 3×3-grid. A pixel in black is encoded as 1, and a pixel in white as 0. Number 1 is encoded as the binary text 010 010 010 For the computer, it becomes the binary sequence 010010010. By adopting this simplification, we ignore the importance of geometry.

1.2 The architecture of the boolean classifier

Our first abstraction of a brain was that it is a conglomerate of neurons that are modeled as boolean gates and that have among them a lot of interconnections. In important cases, such as in the retina and the neural cortex, neurons are located in layers. These simple ideas will allow us in a future to propose a general architecture for a classifier. At present, we will concoct a simple solution for the 1-else-0 classifying task. Its coarse architecture can be seen in figure 2.

Let us pass now to the concrete circuit that solves our task.

1.3 The Boolean Classifier

If we look at the image of number 1 and its binary representation, we can infer that to discriminate images all we need to do is to check that pixels must be 1 in some specific sites and 0 in the rest. And this is certain for whatever task. That is why we can automate the discriminating task by considering boolean circuits. We will use boolean gates to design our discriminator. We choose the family composed of AND, OR, and NOT. The AND gate has

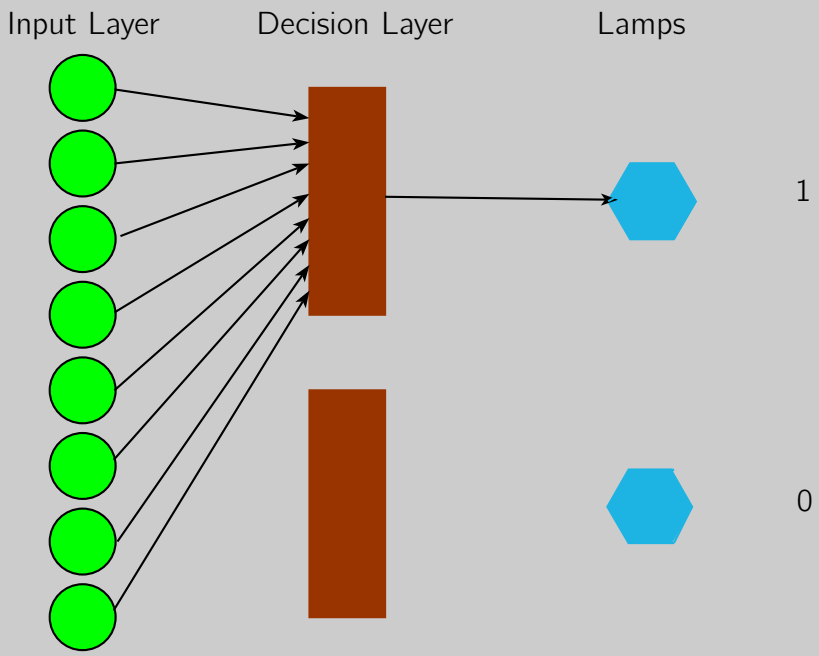


Figure 2: The 3×3 -grid has 9 pixels that are represented by a binary sequence with 9 bits. Each pixel is read by a dedicated neuron of the first or input layer. The overall machine has two outputs that we can imagine as lamps. The upper lamp fires when a 1 has been detected. The lower lamp fires when a zero is detected. To shine, each lamp receives a bit with a value of 1 or true. The intermediate layer of neurons is divided into two modules, the first must activate the upper lamp, and the second the lower one. This architecture has a virtue: it is very simple and intuitive. Its drawback: it is too costly because it has two modules that sound like a repetition. So, other architectures might be explored.

two inputs and one output. Its output is 1 or TRUE if and only if both inputs are TRUE. And OR gate has two inputs and one output. Its output is 0 or FALSE if and only if both inputs are FALSE. Thus, it outputs 1 when at least one input is 1 or TRUE. The NOT gate has one input and one output. It outputs FALSE when the input is TRUE and TRUE when the input is FALSE. These operators are represented so: AND by $\wedge$, OR by $\vee$ and NOT by $\sim(p)$ (tutorialspoint [14] 2023; Xin [16] 2023).

Keep in mind that our images of numbers are perfect but only in this introductory example. The upper module of our decision layer must check whether the input is the binary representation of the image of 1 that is 010010010. Therefore, we must devise a boolean circuit that outputs one when this sequence arrives and zero otherwise. Let us denote by n_i the truth or binary value of gate i of the input layer, where numbers run from 1 to 9.

The task of recognizing 1 is represented by a boolean formula. All we have to do is translate the sequence 010010010 into a Boolean formula in which zero corresponds to false, a negation, and one corresponds to true, an affirmation:

$\sim n_1 \wedge n_2 \wedge \sim n_3 \wedge \sim n_4 \wedge n_5 \wedge \sim n_6 \wedge \sim n_7 \wedge n_8 \wedge \sim n_9$

that says that to output 1, the value of n_1 must be false, a negation, and that of n_2 must be true, an affirmation, and that of n_3 false, ..., and that of n_9 false.

To translate this formula into a logic circuit, we must take into account that gates AND and OR are binary, admitting two inputs, while NOT is uni-ary, admitting one input. Thus, we rewrite it just as

$[((\sim n_1) \wedge n_2) \wedge$
$((\sim n_3) \wedge (\sim n_4))] \wedge$
$\{[(n_5 \wedge (\sim n_6)) \wedge$
$((\sim n_7) \wedge n_8)] \wedge$
$(\sim n_9)\}$

Its graphic representation is in figure 3.

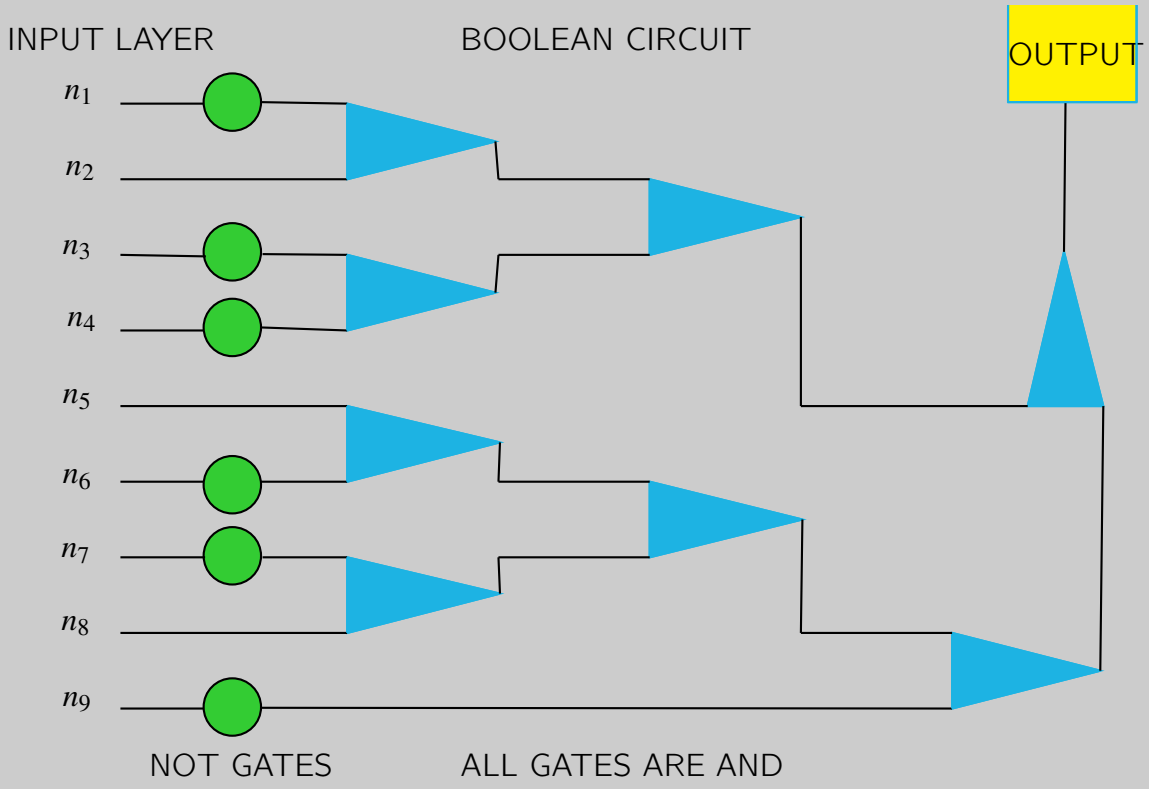


Figure 3: This boolean circuit consists of NOT and AND gates. It responds with TRUE or ONE if and only if the input layer receives the binary string 010010010. For any other sequence, the answer is FALSE or ZERO. We interpret 0 as NOT-KNOWN or UNKNOWN.

6 Exercise. *Verify that this circuit fulfills the predefined task.* *Answer*

7 Challenge. *Write a program in JS that calculates the truth table of the cited formula to verify that it answers 1 when and only when the input is 010010010.*

8 Exercise. *Build another circuit that implements the same function as the previous example. Infer that many circuits solve a given spec function.* *Answer*

9 Exercise. *Design a circuit that outputs ONE or TRUE when it detects a zero. With this, we complete the boolean classifier that lights the upper light when ONE is received as input and the lower one when the input is a ZERO.* *Answer*

10 Exercise. *Which is the spec function of the complete system that detects a 1 else a 0?* *Answer*

2 NOISE AND VARIABILITY

In this section, we will explore the following idea: *Neurons are stable boolean circuits that can correctly make classifications despite noise and variability*. **We must convert this idea into a scientific program that builds abstractions, test them, measure variables, and extract conclusions**. Let us begin.

We have considered that our images are perfect and noise-free. What could happen if we introduce noise and variability?

In our black-or-white frame, noise changes the color of a pixel at whatever site in the grid. A 0 is flipped into a 1, or a 1 into a 0. On the other hand, variability appears in handwritten texts when each person draws signs in different forms. The ensuing pattern recognition problem is so burdensome that from time to

time everyone makes letters that nobody can understand, even the person who wrote them.

This poses a terrible engineering problem:

11 Challenge. *Design the image representation of the set of numbers 0 and 1 that allows the maximal discriminating power when noise and variability are allowed. The Author guesses that the usual representation, which was also adopted in the text, is optimal in that regard: it has the minimum number of "not-known" cases and of wrong classifying. And what about the whole set of 10 digits?*

Let us pass now to propose some solutions to the problem of noise that also involves variability.

2.1 Solution with boolean circuits

Let us show that with boolean circuits one can implement a discriminating circuit for noiseful images. The best way to do this is by developing a method to build the circuit. Let us do it.

We already know how to make a circuit that detects a specific bitmap. So, let us put together all images that are classified as a given pattern, say, those that represent 1. Thus, for each bitmap of the resulting class, we build a recognizing module and join them all together with OR gates. So, if any variant is found, the whole subsystem will respond TRUE.

12 Exercise. *Would you recommend using the just proposed solution to deal with real data, such as those of hand-written letters, whose chars are encoded in* 28×28 *pixel images?* Answer

Since boolean gates offer no real possibility for the treatment of noise, let us consider Turing Machines or ordinary computing. But, beware, the primordial idea of boolean gates cannot be abandoned because anyway every recognizing problem can be

formulated and solved in terms of boolean operations, to be or not to be.

2.2 Solution with Turing Machines

The recognizing problem in the presence of noise has an immediate answer with Turing Machines or ordinary computing as follows:

Suppose that the task is to discriminate between images whose bitmaps are 010010010 and 111101111 that correspond to 1 and 0 respectively. Noise produces 110111101. Must it be interpreted as a 0 or as a 1? To decide, we calculate the distance of the observed sequence to each target, find the minimum one, and the corresponding target is singled out as the correct interpretation. Now, how do we calculate the distance?

13 Definition. *The Bit-by-Bit distance (BbB or BBB distance) between two binary sequences or strings of equal length is the number of sites or bits in which the two sequences differ.*

14 Example. *We are using binary strings that represent bitmaps, binary descriptions of an image. The BbB distance counts the number of sites in which the observed pattern differs from the target:*

```
Target 1, 010010010, one
Observed  110111101
Distance  *  * **** 6


Target 2, 111101111, zero
```

```
Observed  110111101
Distance    * *  *  3
```

```
The minimum distance is 3,
and we decide that
the observed image represents a zero.
```

Thus, the distance of the observed or test string to the bitmap of 1 is 6, and that to the bitmap of 0 is 3. Hence, we decide that the test string represents a 0.

15 Exercise. *Implement the aforementioned idea of the BbB Distance in a JavaScript program.* Answer

16 Criticism. *Reasonable as the classifying criterion of the BbB Distance might be, it is not unconditionally satisfactory. Consider, for instance, the following possible result:*

```
101000010 Random Sequence
010010010 ONE
Distance to ONE = 4
```

```
101000010 Random Sequence
111101111 Zero
Distance to Zero = 5.
```

```
It IS A ONE
```

We see a binary sequence that is classified as one. But, after drawing it, my intuition prefers to classify it as a zero:

```
101
000
010
```

17 Challenge. *Study yourself to detect conflictive situations and develop and implement a criterion in JS that matches your intuition.*

The rapid solution of the problem of noise by the BbB distance prompts a question: why are NNs important given that ordinary computing is so simple, intuitive, and elegant?

The problem is complexity. It means that there is no rapid solution for all cases. Instead, there are many cases in which conflictive situations irremediably arise in great quantities whose solution demands separate study. It is here where NNs enter: they do not need to be programmed to solve every specific case. Instead, they receive all cases and then they learn to solve them. Anyway, let us keep an eye on the assessment of the performance of NNs: Do they perfectly solve every problem, and at which price?

In the meantime let us consider now the biological solution to the noise-proof recognition problem.

2.3 Abstracting the neuron

Neurons are complex, as in structure as in function. Some few characteristics have been abstracted in the models that have propelled Artificial Intelligence. The first is their stability against noise: if a signal is weak, it fades away and is forgotten, but if the signal is above a given threshold, it is propagated further. This property allows us to devise **Boolean Neurons** that are modified boolean gates and that are noise-proof. One must recover the usual logical properties of gates but additionally, circuits must be stable against noise.

18 Example. *Let T be a number somewhat less than 1. An OR-Neuron is a gate with various inputs. If each input is less than T, the neuron makes nothing, a behavior that is interpreted as a ZERO or FALSE or UNKNOWN. But if at least one of the inputs surpasses T, the gate fires and responds 1 or TRUE.*

19 Exercise. *Define an AND-Neuron. Answer*

20 Exercise. *Define a NOT-NEURON. Answer*

With these constructions, we see that the study of neurons is promising.

2.4 The perceptron

According to materialism, we think with the brain. This is composed of neurons. **Thus, The Scientific Method predicts that a machine made of fair abstractions of the neuron must be able to think.**

A trustworthy abstraction of the properties of the neuron was called the **perceptron** and was introduced by Frank Rosenblatt in 1958 ([13]).

The properties of neurons that are abstracted by the perceptron are:

- Stability against noise: if a signal is weak, it fades away and is forgotten, but if the signal is above a given threshold, the signal is propagated further.
- Neurons can assign different weights to different channels. Say, a stimulus near the body of the neuron could be more effective than another one with the same force but in a distant region.
- Neurons can activate else inhibit other neurons.

A perceptron can naturally be interpreted as a machine that makes decisions. See the figure 4. It might receive binary inputs that say whether a certain factor exists, 0 when not, 1 for yes. These inputs are multiplied by a number that represents the importance. The results of the multiplications are added together. If the sum surpasses a certain threshold, the decision is affirmative else it is negative. From this, we immediately infer that neurons also can make decisions. Thus, every organism with just one neuron is eventually capable of making decisions on its own.

This is a good point for materialism, whose purpose is to boost the belief that we think with the brain. But this is false: we think with the spirit (Rodriguez [12] 2023).

21 Example. *One must decide whether to continue reading an article. The important factors are* `easy reading`, `interesting`, `good graphics`, `instructive`, `abundantly cited`, *and* `written by a chat`. *The article at hand has the following traits:* $[0,1,0,1,0,1]$. *A notation that means: the article is not easy to read, it is interesting, lacks graphics, is very instructive but not cited, and has been written by a chat. The array of importance values of these factors is respectively* $[0.2,0.8,0.6,0.9,0.2,-2]$ *meaning that to be instructive is the most important quality of an article under the condition that is written by a human being because a chat is not responsible for what it says. The decision shall be affirmative if the overall value surpasses 0.8. Shall one continue reading?*

We multiply each factor by its weight and add the results:

$0\times 0.2+1\times 0.8+0\times 0.6+1\times 0.9+0\times 0.2+1\times(-2)=0+0.8+0+0.9+0-2=-0.3$

Since the sum is less than the threshold, the perceptron advises us to stop reading. We see that irresponsibility is the dominant factor that drives us to stop reading.

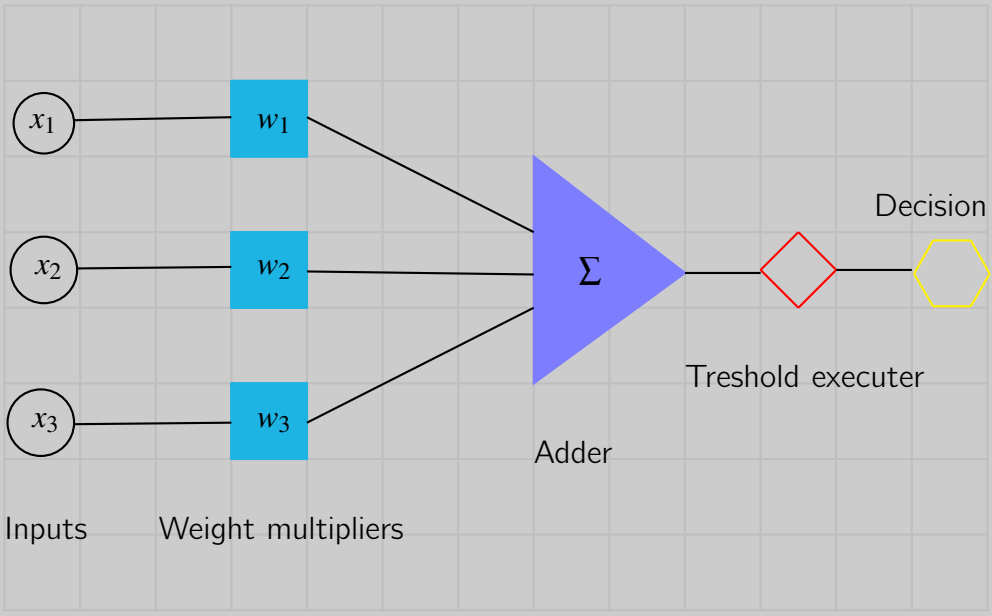


Figure 4: This is a perceptron. It is an abstraction of a neuron that can naturally be interpreted as a machine that makes decisions. Suppose we need to decide whether to continue reading an article. We have as input to the perceptron the presence or absence of some factors, x_i, say, the article is critical to building a worldview, and it is interesting. Factors have different degree of importance to influence the decision. This is modeled by weights w_i. So, inputs are multiplied by their weights and their multiplied values are added together, $s = \sum w_i x_i$. To enhance the recognizing power of the perceptron, a positive constant b is added to the sum. We get a great sum $S = s + b$. The threshold executor checks the resultant value S against the threshold T. If $S \geq T$ the decision is affirmative, else it is negative (edited for version 3).

With a small correction, perceptrons can be made into universal noise-proof gates, with the capability to simulate any other one. To achieve this, we add a value b to the sum (correction for version 3). In that way, we get $S = \sum w_i x_i + b$ as the input to the threshold executer that checks the value of S against the threshold T: if $S \geq T$ the decision is affirmative, else it is negative. In the literature, the shifting value b is called the **bias**.

22 Exercise. *Prove officially that a perceptron calculates a function.* *Answer*

23 Example. *Let us encode the perceptron in JS.*

To enable the possibility to run our program in any browser and cell phone, we embed the JS code inside an HTML template:

```
<!DOCTYPE html>
<html>


<head>


<meta charset="utf-8" />
<title>Perceptron.html </title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="generator" content="Bluefish 2.2.12, Visual Studio Code">
<meta name="author" content="jose">
<meta name="date" content="2023-06-23T13:13:00-0500">
<meta name="copyright" content="Licence = unconditional">
<meta name="keywords" content="Perceptron, AI, artificial intelligence">
<meta name="description" content=" This program calculates the output of a Perceptron.">
<meta name="ROBOTS" content="NOINDEX, NOFOLLOW">
<meta http-equiv="content-type" content="text/html; charset=UTF-8">


</head>


<body>


<div id="myDoc"></div>




<script>
var
message = "<h1>Perceptron <br> Shall one continue reading? </h1>";
bias = 0.2; //a small temptation to continue reading
factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
inputs = [0, 1, 0, 1, 0, 1];
weights = [0.2, 0.8, 0.6, 0.9, 0.2, -2];
threshold = 0.8;
advice = "";
message += " <h3><br>Parameters <br> bias = " + bias
+ " //a small temptation to continue reading"
+ "<br> factors: " + factors
+ "<br> inputs = " + inputs
```

```
+ "<br> weights = " + weights
+ "<br> threshold = " + threshold;


let sum = bias;
for (let i = 0; i < inputs.length; i++) {
sum = sum + inputs[i] * weights[i];
}


//Print real numbers with two decimals only.
sum = 1.0 * Math.round(sum * 100) / 100;
message += "<br> sum = " + sum + "</h3> <br>";


if (sum > threshold) {
advice = "<h1>sum > threshold <br> Advice: Continue reading!</h1>";
} else {
advice = "<h1>sum < threshold <br> Advice: Stop reading!</h1>";
}
message += advice;
document.getElementById("myDoc").innerHTML = message;
</script>


</body>


</html>
```

24 Exercise. *Run in a browser the program* `Perceptron.html`. *Check that everything is in order. Your gentleness would make you ready to run other programs that accompany this article. To that aim, you can copy the code to the clipboard, paste it into a word processor, and save it with the name* `Perceptron.html`. *To run it: over a new tab in your browser, press Control + O to call a dialog for opening files, select the program and it will run automatically. You can also download the zipped file* `evolNNintro.zip`, *unzip it, select the program, and run it over a browser. Please, stop being frightened and experiment until you can see the output of the program.*

25 Challenge. *Run in a browser the program of a perceptron by W3Schools ([15], 2023). Compare it with our version.*

26 Exercise. *Coding is difficult. That is why tools exist to ease work. For instance, every person, from beginners to experts, appreciates editors with colorful rendering and syntax discrimination. One can use, for instance, Visual Studio Code, Eclipse, or NetBeans with the appropriate version for the language in use.*

So, choose one and open this file to see how good it looks. To develop a program, one writes it in the editor and saves it. Next, one opens it in a browser (try CNTRL + O), which runs it automatically after opening it. JS is particularly difficult because many execution errors are not published. Learning to develop programs is important because one must ask oneself: Why do I commit so many bugs or errors, very stupid in general? Next, it might be intelligent to ask: given that our genome is software, why does everyone believe that we are a product of Evolution when the mandatory abundance of bugs is nowhere?

Let us pass now to justify an important claim.

2.5 The perceptron generalizes boolean gates and neurons.

Perceptrons are important because they generalize boolean gates and so are useful for pattern recognition.

27 Example. *The OR-GATE can be simulated by a perceptron as follows: Two inputs that admit boolean values, 1 else 0. Two weights, both equal to 1. Bias equal to 0. The threshold value is equal to 1. If at least one input value is TRUE, the sum that arrives at the threshold executer is equal or greater than the threshold: the perceptron outputs TRUE. If both input values are zero, the shifted sum is zero, and the perceptron outputs FALSE because the threshold has not been surpassed.*

To get an OR-NEURON that is stable against noise, one can change the threshold value in the previous OR-GATE to 0.8 instead of 1. With this new value, the perceptron responds TRUE even when the signals are attenuated. But it does not respond TRUE in some other cases if only the noise is moderate. Because if the noise is extreme, zero values might be converted into 0.97 and so false positives may appear.

28 Exercise. *Specify the values of the parameters of a perceptron to simulate an AND gate and neuron.* Answer

29 Exercise. *Specify the values of the parameters of a perceptron to simulate a NOT gate and neuron.* Answer

30 Example. *In our real life, complexity can be characterized by a simple receipt:* `do this but not that`. *Natural neurons can directly deal with this type of task (Yoder [18], 1989). Let us show that a single perceptron can simulate a boolean circuit, without noise, that encodes* `do this but not that`.

A perceptron can simulate a boolean circuit of the form $p \wedge \sim q$: it must have two inputs, one for p with weight 1, another for q with weight -1, $bias = 1$, and a threshold of 2. The sum is $S = p - q + 1$. The perceptron will output 1 or TRUE when $S \geq 2$. This happens if and only if $p = 1$ and $q = 0$, meaning that p is true but q is false.

31 Exercise. *Prove that a single perceptron can implement the spec function* `do this but not that` *and that is stable against noise.* Answer

32 Challenge. *Devise your personal simulations of boolean and neural gates over a perceptron. Compare them with our solutions and with that of the literature (Banoula, [2], 2023). How would you decide which is the best?*

Implication for neurology. Perceptrons have an exterior appearance composed of inputs+weights+bias. It is the same everywhere but because of inner differences, one perceptron might fulfill a quite different function from another. It is alluring to suggest that our brain is just the same: it is filled in neurons that look the same to an exterior observer but that are quite different in their functions in the brain network. To convert this nice idea

into Science seems to me extremely difficult: while a significant population of neurons is composed of some few types, the complete population of neurons can now be classified by thousands of types (Kwon [7] 2023). What are their roles and why are necessary so many types?

We have seen that perceptrons are indeed powerful in isolation. What can they do when they work in an ensemble?

2.6 Neural Networks (NN)

Around the turn of the XIX century, Golgi and Cajal stained nerve cells and were able to distinguish the synapses where a neuron communicates something to another (Glickstein [5] 2000). **This discovery leads to the idea that neurons form circuits.**

Paralleling this observation, and given that perceptrons are universal neural gates, our abstractions lead us to the following definition:

33 Definition. *A NEURAL NETWORK (NN) is a circuit made of perceptrons. The literature loves to highlight that this type of NN is Artificial.*

Perceptrons are directional from inputs to outputs. Thus, they can naturally be joined or composed so, a NN has an entrance that receives inputs and an exit that produces outputs. Guided by the structure of boolean circuits and of the brain anatomy, say of the retina and the neocortex, our circuits would consist of layers and various output lamps that fire when some trait has been discovered in the input data. In this article, we experiment with the simplest NN, with 2 perceptrons or cells apart from those that receive the inputs.

To calculate the output of a network, we need to recognize that a NN calculates a function. So, a network is calculated

according to the rules of function compositing.

2.7 Building simplicity

We already know how to use boolean gates to make a circuit that recognizes a pattern. To devise an NN that does the same thing, we can replace each boolean gate with the corresponding perceptron, an AND-gate with an AND perceptron, and so on. The resulting NN must be stable against noise, i.e., it must recognize the target pattern even when it comes slightly distorted.

34 Challenge. *Test the aforementioned claim.*

Let us depart from this natural procedure. Instead, let us experiment with the possibilities of perceptrons to create simplicity.

35 Example. *Let us devise a NN to recognize the spec function of 1 in the* **0-else-1-problem**.

The bitmap of 1, 010010010, induces the boolean formula or spec function

$\sim n_1 \wedge n_2 \wedge \sim n_3 \wedge \sim n_4 \wedge n_5 \wedge \sim n_6 \wedge \sim n_7 \wedge n_8 \wedge \sim n_9$

This formula is a function: for each sequence of length 9 of boolean values, it outputs either TRUE else FALSE.

Now, since all connectors are AND, instead of 9 AND-perceptrons, we build an AND-perceptron that admits 9 inputs some of which are priorly negated, a fact that is represented by a negative weight plus a convenient bias. And that is all to it. Explicitly:

The spec function 010010010, the bitmap of 1, must determine the parameters of the perceptron:

- The formula has 9 AND inputs. To be TRUE, every site must be TRUE. So, the threshold is 9.
- For each TRUE or 1 in the input, the weight must be 1.

- For each FALSE or 0 in the input, there must be a negation. That of y is $1-y$. So, the corresponding weight is -1.
- The value of the bias is determined so: there are six negations, so we must add 6 to the bias.
- To achieve stability, let us subtract 1.8 from the threshold. This is equivalent to adding 1.8 to the bias. Total bias = 6 + 1.8 = 7.8.

The code is in the program `PerceptronOne.html`. The corresponding JS code, without the HTML covering, looks like this:

```
<script>
    //This perceptron shall recognize the bitmap
    //010010010
    //as the bitmap of the image of 1 in a 3x3 grid.
    var
      message = "<h1>Perceptron <br> "
        + "Do you recognize 010010010 "
        + "as the bitmap of 1? </h1>";

    factors = "Each pixel is a factor, 9 in total. "  ;
    //This is the bitmap of 1
    inputs =   [0, 1,  0,  0, 1, 0, 0, 1, 0 ];
    weights = [-1, 1, -1, -1, 1,-1,-1, 1,-1];
    //To know the value of the threshold:
    //The evaluation of the formula over the spec bitmap
    //shall be equal to 9 TRUE values
    //because everything is TRUE. So, the threshold is 9.
    threshold = 9;
    //The value of the bias is determined so:
    //The sum evaluated over 010010010 must be equal to the threshold
    //Each negation of y is given by -y + 1.
    //Since there is 3 TRUE values in 010010010,  the bias must be 6.
    //To achieve some stability  against noise,
    //let us subtract 1.8 from the threshold, i.e., 0.2 for each bit of the input.
    //This is equivalent to adding 1.8 to the bias.
    //Total bias = 6+1.8 = .7.8.
    bias = 7.8; //
    verdict = "";
    message += " <h3><br>Parameters <br> bias = " + bias
      + " //to deal with negations and stability"
      + "<br>  factors: " + factors
      + "<br>  inputs =  " + inputs
      + "<br>  weights =  " + weights
      + "<br>  threshold =  " + threshold;


    let sum = bias;
    for (let i = 0; i < inputs.length; i++) {
      sum = sum + inputs[i] * weights[i];
    }


    //Print real numbers with two decimals only.
    sum = 1.0 * Math.round(sum * 100) / 100;
    message += "<br> sum = " + sum + "</h3> <br>";

    if (sum > threshold) {
      verdict = "<h1>sum > threshold <br> Verdict: a 1 is recognized!</h1>";
    } else {
      verdict = "<h1>sum < threshold <br> Verdict: the bitmap es not that of 1!</h1>";
    }
    message += verdict;
    document.getElementById("myDoc").innerHTML =  message;
```

```
</script>
```

36 Exercise. *Run the program* `PerceptronOne.html` *to test the claim that this perceptron correctly classifies the bitmap of 1.*

37 Exercise. *Devise a perceptron to detect a 0 about the* **0-else-1-problem**. *Answer*

38 Exercise. *We have two isolated perceptrons, one to detect the bitmap of 1, and the other that of 0. How must they be assembled into a NN that resolves the* **0-else-1-problem**? *Answer*

We have created a circuit with perceptrons that resolves the **0-else-1-problem**. But until now we have no idea about the stability of its reaction to noise. Say, slightly different bitmaps from that of 1 shall still be classified as ONE, but too different ones must be classified as NOT-ONE. Is that true? To know it, let us use the BbB distance to know the nearness of the bitmaps to the targets in the **0-else-1-problem**. Our problem is that we still have a void:

39 Example. *Let us devise a method to produce all possible bitmaps. Implement it in JS and include a classification of each bitmap either as 1 or 0 according to the BbB distance. Have your program output the bitmaps sorted by distance.*

We need in the first place to determine the number of bitmaps. A $3 \times 3 = 9$ image whose pixels are black or white generates $2^9 = 512$ possible bitmaps or binary sequences. To generate all bitmaps, we list all numbers from 0 to 512, transform each number into binary notation, and refill void spaces with 0.

40 Example. *Number 5 is 101 in binary. Since our bitmaps have 9 bits, we lack 6 chars that must be replenished with 0's. The resultant string is 000000101. This is a bitmap that this procedure would not repeat. The code is in the program* `allBitmaps.html`*, which classifies each bitmap as a distortion of 1 else 0. For that, it uses the BbB distance: the minimum distance decides the identity. At the end of the results, we see a list of all bitmaps sorted by the distance to the closest target, be it the bitmap of 1 else 0.*

41 Exercise. *In the program* `allBitmaps.html`*, all bitmaps were reported as strings. Develop a program that reports them as drawings, in a 3-line format 3 x 3 pixels. Thus, the bitmap 111000111 must look like:*

```
111
000
111
```

It is now natural to say that this bitmap corresponds to 0. Does your common sense agree with the classification offered by the BbB distance? Answer

Once we know how to produce all bitmaps, we can proceed to assess directly the behaviour of our circuit under variability. This is done in the program `CompareTwoClassifications.html`. This program compares the classification of bitmaps according to the BbB distance with that given by our circuit of two perceptrons.

Our results read:

The NN of two perceptrons, the first that detects 1 and the second 0, reacts to tiny noise in due form: when there is a change in only one site, and the BbB distance is 1, all mutated bitmaps were classified correctly, be they deformations of 1 else

0. The space of stability consisted of 10 bitmaps that were correctly classified, corresponding to the original bitmap plus 9 mutants, one for each site in the binary string. Example: Original bitmap of zero = 111101111, mutant Bitmap 111111111 that was correctly classified as zero by Perceptron0 and as unknown by Perceptron1.

42 Exercise. *In the program* `CompareTwoClassifications.html` *change the biases of the two perceptrons to enhance its tolerance without losing reliability. This means a maximal number of matches but not mismatches or errors of classification.* *Answer*

IN HINDSIGHT: By looking at the results of our theory that has been implemented in our programs, let us claim that they are correct. Let us go ahead to our ultimate goal!

Our final purpose is to involve Evolution in our discussions to maximally enhance tolerance but without committing classifying errors. That is why we need populations of our perceptrons and NNs. Our first task is therefore to encapsulate them in JS classes.

2.8 Perceptron as a JS class

We will use perceptrons as the building blocks of NNs. So, let us encapsulate it as a JS class to have the possibility of making many instances to be used as cells.

43 Example. *We transform the program* `Perceptron.html` *into a class and reproduce its output. Our class is the Perceptron. A class is the abstraction of an ensemble of data and their corresponding methods or procedures to process them and publish results. A class is similar to the plan of a house. To convert the plan into a reality, we need an instantiation. The instantiation is done by the modifier* `new` *followed by the name of the class:*

```
let myPerceptron = new Perceptron(0.2, factors, [0, 1, 0,
```

44 Example. *The primordial code of the Perceptron JS class follows,* `PerceptronClass.html`*:*

```
<!DOCTYPE html>
<html>


<head>


<meta charset="utf-8" />
<title>PerceptronClass.html. </title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="generator" content="Bluefish 2.2.12, Visual Studio Code">
<meta name="author" content="jose">
<meta name="date" content="2023-07-05T13:13:00-0500">
<meta name="copyright" content="Licence = unconditional">
<meta name="keywords" content="Perceptron, AI, artificial intelligence">
<meta name="description" content=" This program defines the perceptron class with getters and calculates its output.">
<meta name="ROBOTS" content="NOINDEX, NOFOLLOW">
<meta http-equiv="content-type" content="text/html; charset=UTF-8">


</head>


<body>


<div id="myDoc"></div>




<script>
// Declaration of the class
class Perceptron {
//Declaration of the parameters of the perceptron
bias;
factors;
inputs;
weights;
threshold;



//This definition of the class is similar to the drawing of a plan for a house.
//It is not the house, it is just a plan.
//With a plan, many houses can be made.
//With a class many instances can be created,
//one by one,
//thanks to the constructor method.
//When one instance is created,
//all parameters can be initialized at once:
constructor(bias, factors, inputs, weights, threshold) {
this.bias = bias;
this.factors = factors;
this.inputs = inputs;
this.weights = weights;
this.threshold = threshold;
}


//Methods to retrieve information
get bias() {
return this.bias;
}


get factors() {
return this.factors;
}


get inputs() {
```

```
return this.inputs;
}


get weights() {
return this.weights;
}



// Summ Getter
get sum() {
return this.calcSum();
}
// Method
calcSum() {
let sum = this.bias;
for (let i = 0; i < this.inputs.length; i++) {
sum = sum + this.inputs[i] * this.weights[i];
}
return sum;
}



//Threshold getter
get threshold() {
return this.threshold;
}


*getData() {
yield this.bias;
yield this.factors;
yield this.inputs;
yield this.weights;
yield this.threshold;
}


}//End of class



var
message = "<h1>Perceptron <br> Shall one continue reading? </h1>";
/*
bias = 0.2; //a small temptation to continue reading
factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
inputs = [0, 1, 0, 1, 0, 1];
weights = [0.2, 0.8, 0.6, 0.9, 0.2, -2];
threshold = 0.8;


*/


const factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
let myPerceptron = new Perceptron(0.2, factors, [0, 1, 0, 1, 0, 1], [0.2, 0.8, 0.6, 0.9, 0.2, -2], 0.8, -1);




message += "<br> threshold = " + myPerceptron.threshold;
// document.getElementById("myDoc").innerHTML = message;


message += " <h3><br>Parameters <br> bias = " + myPerceptron.bias
+ " //a small temptation to continue reading"
+ "<br> factors: " + myPerceptron.factors
+ "<br> inputs = " + myPerceptron.inputs
+ "<br> weights = " + myPerceptron.weights
+ "<br> threshold = " + myPerceptron.threshold
+ "<br> <br> All data = " + [...myPerceptron.getData()];


document.getElementById("myDoc").innerHTML = message;



message += "<br> <br> sum = " + myPerceptron.sum + "</h3> <br>";
```

```
document.getElementById("myDoc").innerHTML = message;


let advise = "";
if (myPerceptron.sum > myPerceptron.threshold) {
advice = "<h1>sum > threshold <br> Advice: YES!</h1>";
} else {
advice = "<h1>sum < threshold <br> Advice: NO!</h1>";
}
message += advice;
document.getElementById("myDoc").innerHTML = message;
</script>



</body>


</html>
```

45 Exercise. *Run this program* `PerceptronClass.html`.

Program `PerceptronClass.html` is as yet not useful for Evolution because the values of every instance of the class are frozen.

46 Example. *Let us enable the Perceptron class with the possibility to modify its values. This is done by the following code,* `PerceptronClass2.html`:

```
<!DOCTYPE html>
<html>


<head>


<meta charset="utf-8" />
<title>PerceptronClass2. </title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="generator" content="Bluefish 2.2.12, Visual Studio Code">
<meta name="author" content="jose">
<meta name="date" content="2023-07-08T13:13:00-0500">
<meta name="copyright" content="Licence = unconditional">
<meta name="keywords" content="Perceptron, AI, artificial intelligence">
<meta name="description"
content="This program defines the perceptron class with setters and getters and re-calculates its output.">
<meta name="ROBOTS" content="NOINDEX, NOFOLLOW">
<meta http-equiv="content-type" content="text/html; charset=UTF-8">


</head>


<body>


<div id="myDoc"></div>




<script>
```

```
//================================
//====== The perceptron class ====
//================================


// Declaration of the class
class Perceptron {
//Declaration of the parameters of the perceptron
bias;
factors;
inputs;
weights;
threshold;


//This definition of the class is similar to the drawing of a plan for a house.
//It is not the house, it is just a plan.
//With a plan, many houses can be made.
//With a class many instances can be created,
//one by one,
//thanks to the constructor method.
//When one instance is created,
//all parameters can be initialized at once:
constructor(bias, factors, inputs, weights, threshold) {
this.bias = bias;
this.factors = factors;
this.inputs = inputs;
this.weights = weights;
this.threshold = threshold;


}


// GETTERS: methods to retrieve information


get bias() {
return this.bias;
}


get factors() {
return this.factors;
}


get inputs() {
return this.inputs;
}


get weights() {
return this.weights;
}



// Summ Getter
get sum() {
return this.calcSum();
}
// Method
calcSum() {
let sum = this.bias;
for (let i = 0; i < this.inputs.length; i++) {
sum = sum + this.inputs[i] * this.weights[i];
}
return sum;
}



//Threshold getter
get threshold() {
return this.threshold;
}



get decision() {
return this.calcDecision();
```

```
}



calcDecision() {
let d = -1;
if (this.sum > this.threshold) {
d = 1;
} else {
d = 0;
}
return d;
}




*getData() {
yield this.bias;
yield this.factors;
yield this.inputs;
yield this.weights;
yield this.threshold;
}


// Summ Getter
get message() {
return this.getMessage();
}
// Method
getMessage() {
let message = "<h1>Perceptron <br> Shall one continue reading? </h1>"
+ " <h3><br>Parameters <br> bias = " + this.bias
+ " //a small temptation to continue reading"
+ "<br> factors: " + this.factors
+ "<br> inputs = " + this.inputs
+ "<br> weights = " + this.weights
+ "<br> threshold = " + this.threshold
+ "<br> <br> All data = " + [...this.getData()];
return message;


}



//SETTERS, methods to redefine information


set bias(x) {
this.bias = x;
}


set factors(x) {
this.factors = x;
}


set inputs(x) {
this.inputs = x;
}


set weights(x) {
this.weights = x;
}
set threshold(x) {
this.threshold = x;
}
set decision(x) {
this.decision = x;
}


}//End of class
```

```
//The Evolutionary Environments for Neural Networks


/* Factors to decide whether to continue reading.
bias = 0.2; //a small temptation to continue reading
factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
inputs = [0, 1, 0, 1, 0, 1];
weights = [0.2, 0.8, 0.6, 0.9, 0.2, -2];
threshold = 0.8;
*/
var
factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
myPerceptron = new Perceptron(0.2, factors, [0, 1, 0, 1, 0, 1], [0.2, 0.8, 0.6, 0.9, 0.2, -2], 0.8, -1);
message = myPerceptron.getMessage();
message += "<br> <br> sum = " + myPerceptron.sum + "</h3> <br>";


let advise = "";
if (myPerceptron.sum > myPerceptron.threshold) {
advice = "<h1>sum > threshold <br> Advice: YES!</h1>";
} else {
advice = "<h1>sum < threshold <br> Advice: NO!</h1>";
}
message += advice;
document.getElementById("myDoc").innerHTML = message;


//==========REDEFINITION=============


message += "<br> <h1> REDEFINITION</h1> <br> ";


//Some values are redefined
myPerceptron.threshold = 0.0001;
myPerceptron.inputs = [0, 0, 0, 0, 0, 0];
myPerceptron.inputs[0] = 1;


message += myPerceptron.getMessage();


message += "<br> <br> sum = " + myPerceptron.sum + "</h3> <br>";



if (myPerceptron.decision) {
advice = "<h1>sum > threshold <br> Advice: YES!</h1>";
} else {
advice = "<h1>sum < threshold <br> Advice: NO!</h1>";
}
message += advice;
document.getElementById("myDoc").innerHTML = message;



</script>



</body>


</html>
```

47 Exercise. *Run this program* `PerceptronClass2.html`.

The difference between these two versions is the following: in the first program we included all the necessary machinery to retrieve information from any instance of the class. This was done with getters. But to modify the information content, we need

setters. The possibility to change values is tested and results are shown in the output of the program. This enables Evolution, which is change over change. Possibly, we were exaggeratedly legalist and the code might be much more simple.

With the Perceptron class, we can make many instances of it to have many cells forming a circuit or NN. And since we need populations of NNs to program Evolution, we must encapsulate an NN as a JS class.

2.9 The code for a NN as JS class

Let us recast the program `CompareTwoClassifications.html` which depicts an NN with two perceptrons in the language of JS classes. This is done in the program `CompareTwoClassifications2.html`. Observe that the declaration of the NN as a class involves its architecture. This implies that we have not one single declaration for all NNs but that we must define a specific class for each architecture. That is why we should understand very clearly the modus operandi of the code.

The architecture of the present NN consists of: 9 inputs correspond with the 9 bits of each bitmap. These enter as raw data to each one of the two cells of the only layer that takes final decisions. So, this layer is called the decision layer. The first perceptron outputs 1 when ONE is detected. The second outputs 1 when ZERO is detected. We interpret output 0 as NOT-KNOWN.

48 Exercise. *Add some statistics to the program* `CompareTwoClassificati` *to see the role of the degree of tolerance over reliability. Test the conclusion of the Author:* **the employed NN can reach 260 matching without mismatches. This is our top that will taken as perfection.** *Answer*

We need these results to compare them with those of Evolution that is researched in the following section.

3 LEARNING

A Neural Network (NN) is an abstraction of the neural system. NNs have trapped the attention of the whole world because they not only can execute calculations but can also be joined to an external facility that together simulate learning.

Let us pay attention to a very simple fact that will help us avoid messing up concepts: An NN is a special circuit that always responds to any input. So, an NN calculates a mathematical function that to any allowed given input associates the resultant output. This function is the **generated function**. An NN can exist by itself, is frozen, and cannot learn. But, we can fabricate an environment for the NN, in which learning naturally occurs.

The idea of learning is indeed simple. The generated function and the spec functions may differ and the difference or error can be measured. The following is a common way, but not the unique -see below:

$$Error = \sum_{inputs}(f_{spec}(input) - f_{generated}(input))^2$$

In general, the error is not nil. **Learning happens when one makes changes that systematically diminish the error.** Thus, neural networks are faced with a test in which a subset of patterns is given for classification. If the answers match the correct ones that are already known, no change is made, but if a mismatch is detected, changes are proposed to improve performance, i.e., to diminish in future tests the total error of the NN.

To minimize the error, we are dealing with the task of finding the minimum of a multidimensional eggshell or paraboloid. The error is called in the literature the **cost function**, an expression taken from the jargon in optimization problems.

Many methods exist to find good approximate solutions to optimization problems. For the NNs, the following is the standard.

3.1 Back Propagation Algorithm

When a quadratic measure of error can be used, minimizing it corresponds to finding the bottom of a paraboloid. This minimum can be approached in learning when tiny changes are made in the direction in which water flows to reach the bottom of the eggshell. This is the so called *Back Propagation Algorithm*. This methodology is intuitive, simple, and -in my opinion- extremely efficient.

49 Exercise. *Enjoy the excellent video of 3Blue1Brown, ([1] 2018) about the Back Propagation method.*

50 Exercise. *Read the wonderful exposition with embedded interactive demonstrations about NNs by Nielsen ([8] 2018). The backpropagation algorithm can be found in Chapter 2. An informal and understandable proof that NNs can approximate any function is found in Chapter 3.*

51 Challenge. *In the usual approach to NNs, one must minimize a paraboloid. That is an easy task because it has only one minimum. But everyone says that NNs are very complex. Dismantle this contradiction.*

In a process of learning, one compares the objective with the actual state of the system and one executes changes intending

to minimize the discrepancy between the two. This is an optimization problem. Good approximations to this type of problem can be given along diverse methods. Ours is Evolution.

3.2 Evolutionary Environments for Neural Networks (EENN)

Our Evolutionary implementation of learning consists of putting a population of NNs to suffer mutation and recombination that produces changes at random. The instance that best does the predefined task is selected for cloning to form the next generation. And so on, until matching is perfect or a plateau or a pseudo-cycling process appears without further evolution.

It is interesting that the backpropagation method is inspired by direct geometry and implements deterministic changes. By contrast, Evolution uses random changes and their recombinations and those that improve function are selected for reproduction. Can Evolution beat other methods to minimize the error of an NN?

No. If that were the case, that would be unexpected. The problem is to find the minimum of a paraboloid for which a very intuitive and efficient method already exists -the back propagation algorithm. Nevertheless, the application of Evolution to NN learning was well established by 1993 (Radcliffe ([11] 1993, Yao [17] 1993) to optimize connection weights of a NN, its architecture and also its learning rules. Thus, this theme is one of the most fructiferous proposals in the field because the last two objectives are outside the scope of the back propagation algorithm. Here we will learn over a toy model the modus operandi of this approach. It is simple, natural and has been deeply studied (Kenneth, [6] 2019).

Besides, and what is most important for us, this approach blends two important items of modern Science: Evolution and NN, both of which are related to our understanding of the nature

of man. Thus, our task is to study this approach to chase high-level questions. Let us begin.

52 Definition. An Evolutionary Environment for Neural Networks (EENN) *is a population of neural networks that recurrently suffer blind changes and their recombination and whose fitness or performance to optimize a given aim determines the procedure of cloning.*

Let us pass now to the simplest application of Evolution: we have designed an NN that decides whether an image corresponds to ONE else ZERO. We have found a way to extend the stability of the NNS by changing the bias of each perceptron or cell. Let us use Evolution to answer a simple question: how and how much can the bias be moved to maximize the performance and reliability of the NN?

Let us review the algorithm to simulate Evolution that is used by our EENN.

3.2.1 Implementation

Our problem is to have Evolution find the values of the biases of two perceptrons that maximize the capability of the NN to recognize a ONE or a ZERO even when their images are distorted.

Our algorithm begins with a population of NNs generated at random. Next, the population is subject to a recurrent cycle that consists of

1. Assessment of the score or fitness of each NN which is given by its correct matchings minus a chastise due to its classification errors. Every NN is tested over all bitmaps and the corresponding score is given, which directly determines its fitness.

2. The fittest is cloned to generate a new population with the same number of individuals.

3. The population is subject to mutation: some changes are made to the clones.

4. The population passes under recombination or interchange of information.

What do we expect from our EENN? Since the fittest is the only one that reproduces, the population is expected to learn to correctly classify the bitmaps as generations run over and over. The corresponding program is `ENN1.html`. It is enclosed in the accompanying zipped file. All programs can be run in a browser once this file has been unzipped. In the cell phone, everything flows by instinct.

We must decide before continuing if we will tolerate errors that happen when the NN classifies an image as ONE, but the BBB distance says that it is ZERO, or vice versa, or when the two perceptons contradict each other -an event that does not happen.

53 Example. *If the BBB distance says that the bitmap is 1 but the NN says that it is 0, we have a mismatch. This means that the perceptron 1 answered 0 (UNKNOWN) while the perceptron 0 answered 1 (the bitmap represents 0). If the BBB distance says that the bitmap is 0 but the NN says that it is 1, we get a contradiction. In this case, perceptron 1 answered 1 (it is 1) and perceptron 0 answered 0 (UNKNOWN). If perceptron 1 answers 1 (it is 1) and so does perceptron 0 (it is 0), we get another contradiction. This possibility does not happen. If both perceptrons answer 0 (UNKNOWN), there is no contradiction and no good point.*

So, we test each bias assignation by giving the corresponding NN a test of classifying a given number of bitmaps. Let us define that there is an agreement or match when the NN agrees with

the BBB distance on the classification of a given bitmap and a disagreement or mismatch when not. For us, the BBB is the law. So, if the number of matches is m and that of mismatches is n, the fitness of the NN is

$Fitness = m - n$

This measure suffers from one problem: a match weighs as much as a mismatch. This sounds bad because it says that it is not difficult to compensate for mismatches. However, it is cheap to manufacture errors of classification, chance is more than enough. See an example below. Chance is even capable of producing 256 matches and 256 mismatches over a test with 512 bitmaps.

So, to save our honor, we must avoid mismatches at any cost. Our suggestion is to change the fitness function by:

$Fitness = m - kn$

We have found that $k = 5$ works well for our examples. That was implemented in the program `EvolNN.html`.

54 Exercise. *Run program `EvolNN.html` and verify that, with a high frequency, 20 individuals evolving during a maximum of 70 generations are enough to find the value of the biases of the two perceptrons that render the higher number of matches (260 according to program `CompareTwoClassifications3.html`) and no mismatch. The mutation rate was 0.1 and the recombination rate was 0.2. This shows that the simplest implementation of Evolution works perfectly. Warning: this program and all those that follow are memory-hungry. You need 8 Gigabytes of RAM, to close all aplications, and to use a private dedicated window of your used browser for it. It will run for some 4 minutes before stopping. To stop a program over Firefox, click inside its window and drag down. A dialog will appear with the option to halt the program.*

55 Exercise. *Developing software generates too many bugs or errors. Debugging of programming errors must be public domain. Our directive is that seeing what a program does shall be open to everybody. The implementation in JS demands to keep in memory the outputs that one wants to see and that might be published once the program stops by itself. This task consumes too much memory. So, programs must be run for tiny examples. Thus, modify, circa line 300, the number of generations to be evolved to a maximum of 3. Select line 1383 instead of 1382. Otherwise, the program may cause your operational system to collapse. Turn on the print flags, starting from line 318 and below, one by one. If you turn on one flag, turn off all others. Then, you can check the processing of your selected method.*

56 Exercise. *Add to the program* `EvolNN.html` *the possibility of seeing the number of matches together with the number of mismatches. Verify that mismatches do not happen for a score of 260, just as promised. Else, create and implement a remedy.* *Answer*

57 Exercise. *Prove that by mere randomness, any given NN can fulfill some values of the spec-function although it can commit some errors of classification. Make a program that tests this on the task of finding the optimal values of the bias for the 1 or 0 problem.* *Answer*

58 Challenge. *Based on the previous exercise, we have made a terrible generalization: Randomness is unable to produce high quality NNS. It is terrible because it denies the possibility of randomness to achieve anything possible. Remedy: fabricate a bunch of functions and test the aforementioned generalization. Hint: The BbB distance divides the whole set of bitmaps into two disjoint subsets. In general, every separation of the universal set into two distinct classes also defines a function: to a bitmap*

in one subset, the function associates, say, A, and to bitmaps in the other, B. To solve the challenge, make as many cuts as you want and run the corresponding statistics.

59 Exercise. *Test the power of Evolution to fine-tune only the threshold values of the two perceptrons of the NN that was designed by a human and shown in Program* `EvolNN.html`, *and* `EvolNN2.html`. *Answer*

60 Challenge. *Resolve the following conundrum: It seems to us that the cost of fine-tuning the biases is by far lower than that of tuning thresholds. Is that true or just a sampling effect or, maybe, a side effect of the discontinuity of coding? It this is a real fact, how can it be explained? Advice: to diminish the memory use, study how was the printing of diverse results managed in program* `EvolNN3.html` *and adapt that to program* `EvolNN2.html` *to silence detailed printing. This would speed the velocity of computing some 100 times while keeping memory usage rather low. In that way, enough data can be gathered to run statistics.*

61 Exercise. *Test the power of Evolution to fine-tune only the weight values of the two perceptrons of the NN that was designed by a human and shown in Program* `EvolNN.html`, *and* `EvolNN2.html`. *Compare your results with those of the Author: 100000 generations were not enough to make the task, which was to reach the maximal possible score that we interpret as perfection. Answer*

What could be the cause of our failure to make Evolution succeed? It was possibly due to a lack of recombination with new ideas because the new generation was formed by cloning the champion. To remedy this, we substituted the cloning of the champion with a method in which the score or fitness of

each genome was interpreted as a probability of reproducing by cloning. The behavior was exactly the same. This was done with program `EvolNN4C.html`. In a heavy duty experiment, this program was run with 100 genomes, mutation rate = 0.1, recombination rate = 0.1. Evolution reaches good quality rather soon but it falls down to an absorbent sea of under-mediocrity from which no exit was found along 100000 generations. Thus, **the unmistakably signature of Evolution was found to be persistent undermediocrity**.

62 Exercise. *Can we claim that Evolution cannot reach perfection and that therefore that it is false?* Answer

63 Exercise. *Can we claim that the persistent under-mediocrity, the unmistakably signature of Evolution, is alone sufficient to falsify the Evolutionary Theory?* Answer

64 CORRECTION (Appears only here in version 2). *In the first version of this study we could not make Evolution find the optimal values of the weights of the RN in question. But, working on Volume II of this series we achieved it: by giving the mutation and recombination rates low values, Evolution was able to reproduce the optimal values of the weight vector. Explicitly:*

We ran the program `EvolNN4.html` for two thousand generations on a desktop with 8 Gigs of RAM with the following parameters:

```
POP_SIZE = 500,
MUT_RATE = 0.0001, //per char
REC_RATE = 0.1,
```

The task was to optimize the weight values given that the other parameters were already optimized.

The score of the best grows in leaps throughout the Evolution. And **perfection can be achieved, at least in reference to the best of each generation**, as shown in the following table where the generation in which the leap was made appears:

Generation	score	matches	mismatches	cost
0	64	144	16	500
66	72	172	20	33500
69	112	172	0	35000
143	152	152	0	72000
1996	212	212	0	998500

65 Graduation. We have shown that Evolution can separately tune the optimal values of the biases, thresholds and weights of a very simple RN. Solve the following questions:

1. *Will Evolution be able to maintain the gained perfection?*
2. *Will Evolution be able to adjust all sets of parameters of an RN as they evolve in parallel over time?*
3. *The evolution of the score of the best progresses in jumps. But, in reference to the population in general, is Evolution gradual, or in leaps?*
4. *How variable is the cost to obtain optimality?*
5. *How do you explain that Evolution works when the mutation rate is low and not when it is high?*
6. *We have said that when the mutation rate is low, Evolution works, but when it is high, it does not. What is the barrier that divides the high mutation from the low one?*
7. *What observable restrictions on the supposed Evolutionary Process that gave rise to species are derived from the idea that mutation must be low for it to work?*

8. *What will happen when the task becomes complicated and therefore the RN that solves it becomes very complex by increasing the number of layers and perceptrons per layer and when the connections do not follow any pattern? Will Evolution find optimality?*

9. *Do you consider that our results support the Evolutionary Triumphalism that sees no mystery in the supposed Evolution of man from some primate? An affirmative answer must include an evolutionary procedure to revolutionize artificial intelligence so that it becomes definitively better, more creative, than human intelligence. Otherwise, it would be pure bragging.*

4 CONCLUSION

We have done our best to show that a neuron functions as a noise-tolerant generalized Boolean gate. To achieve this, we have simulated a neuron using a perceptron, introduced in 1958, and a neural network (RN) using a network of perceptrons. A perceptron receives stimuli, each one is assigned a weight, it is added over all the inputs, a value called bias is added that serves as a tolerance regulator, and the total sum is compared with a threshold. If the sum is less than the threshold, nothing happens, the perceptron sends a ZERO, but if it is equal to or greater, the perceptron fires, it sends a ONE. An RN is a natural concatenation of perceptrons. The task that an RN must perform is to correctly classify a series of patterns that it receives as input.

Using an ordinary computer with 8 Gigabytes of RAM, we have shown that Evolution can adjust the isolated parameters of the perceptrons of an RN with 2 perceptrons to maximize hits but without a single classification error. For the first version

of this paper, we were able to have Evolution find the optimal values of the biases and thresholds separately. But we could not specify the weights. But this time, second version, we could. All you have to do is give low values for both mutation (0.0001 per character, per generation) and recombination (0.1). However, for large mutation rates (0.3), performance preferably remained below mediocrity.

Our delayed and conditional success in making Evolution work leads us to believe that the power of Simulated Evolution, in addition to having inherent limitations in terms of excessive cost in memory and time, is also strongly limited by our lack of imagination and tenacity.

5 ANSWERS TO EXERCISES

4, page 6. The zero travels around the border of the 3×3 grid. It is represented by the binary text

```
111
101
111
```

that becomes the binary sequence 111101111.

6, page 11. If we evaluate the formula

$\sim n_1 \wedge n_2 \wedge \sim n_3 \wedge \sim n_4 \wedge n_5 \wedge \sim n_6 \wedge \sim n_7 \wedge n_8 \wedge \sim n_9$

over the string 010010010, we get

$\sim FALSE \wedge TRUE \wedge \sim FALSE \wedge \sim FALSE \wedge TRUE \wedge \sim FALSE \wedge \sim FALSE \wedge TRUE \wedge \sim FALSE$

that reduces to

$TRUE \wedge TRUE \wedge TRUE \wedge TRUE \wedge TRUE \wedge TRUE \wedge TRUE \wedge TRUE \wedge TRUE$

that simplifies to TRUE. For any other string, there is a zero or FALSE in some place and, therefore, the overall formula renders zero or FALSE.

8, page 11. Other circuit results if we apply the associative property of the AND gates:

$p \wedge q \wedge r = ((p \wedge q) \wedge r) = (p \wedge (q \wedge r))$.

More circuits appear if we apply the Morgan's law:

$p \wedge q = \sim (\sim p \vee \sim q)$

9, page 11. The lower module must detect the sequence 111101111 that encodes for 0. This is done by a circuit that realizes the boolean formula

$n_1 \wedge n_2 \wedge n_3 \wedge n_4 \wedge \sim n_5 \wedge n_6 \wedge n_7 \wedge n_8 \wedge n_9$

10, page 11. The spec function of the complete discriminator is: to the sequence 010010010 the first lamp must output 1, and 0 for any other value, while the second lamp must answer 1 to 111101111, and 0 for any other input. The formula that detects 0 must remain separated from that of 1 because each module produces its output. So, the output of the discriminator is a string with two bits, the first is 1 when 1 is detected, and the second is 1 when 0 is detected. Otherwise, they are zero.

12, page 12. We recommend not using boolean circuits to solve the problem of noise with real data because of an exponential explosion of the number of needed gates. Just consider that with images in a grid with $28 \times 28 = 784$ pixels, as usual in real life, we can encode for $2^{784} = \infty$ different bitmaps.

15, page 14. Our code for the BbB Distance will be abundantly used in the following programs. It is the following:

```
// Bit-by-bit distance between two binary bitmaps
//that are 9 chars long.
//For each discrepancy, one is added to the distance.
function distance(s, t) {
let d = 0;
let n = s.length;
console.log("s=" + s + "s.length = " + n);
for (let i = 0; i < n; i++) {
if (s.charAt(i) === t.charAt(i)) {
d++;
}
}
let distance = n - d;
console.log("s = " + s + " t = " + t + " d = " + distance
return distance;
}
```

There are 512 possible bitmaps and of them 256 were classified by the distance method as 1 and the other 256 as 0. Henceforth, the probability of each event is $1/2$.

19, page 16. An AND-Neuron is a gate that has various inputs, say k. Let T be a number a bit less than 1. If the sum of all inputs surpasses kT, the neuron fires and responds 1 or TRUE. Else, it does not fire. Thus, an AND-Neuron is a modified AND-Gate that responds TRUE when the global input is strong enough, a bit lower than k making it a bit stable against attenuating noise.

20, page 16. A NOT-NEURON is a gate that fires and responds with 1 or TRUE when the input is less than a given threshold, but if the input surpasses the threshold, the neuron outputs nothing, ZERO or FALSE or UNKNOWN.

22, page 19. We shall find the mathematical expression of the function calculated by a perceptron.

Let be P a perceptron that accepts n factors, whose inputs are i_j such that $j = 0,...,n$ (not included), with weights w_j, one weight for each factor, bias b, total sum $S = \sum w_j \times i_j + b$ and threshold T, then P calculates the function f:

$$f(S) = \begin{cases} 1 & \text{if} \quad S < T \\ 0 & \text{if} \quad S \geq T \end{cases} \tag{1}$$

28, page 22. Let us specify the value of the parameters of a perceptron to simulate a boolean AND-gate and a boolean AND-neuron.

The AND-gate can be simulated by a perceptron as follows: Two inputs that admit boolean values, 1 else 0. Two weights, both equal to 1. Bias equal to 0. The threshold value is equal to 2. If both input values are TRUE, the shifted sum that arrives at the threshold executer is 2 which matches the threshold: the perceptron outputs TRUE. In any other case, the perceptron outputs FALSE.

To get an AND-NEURON that is stable against noise, one can change the threshold value in the previous AND-Neuron from 2 to 1.8. With this new value, the perceptron responds TRUE even when the signals are attenuated. But it does not respond TRUE in other cases if only the noise is moderate. Because if the noise is extreme, zero values might be converted into 0.97 and so false positives might appear.

29, page 22. To simulate a NOT-GATE, we can exploit the fact that if $y = 1 - x = -x + 1$, if $x = 1$ then $y = 0$, and $y = 1$ when $x = 0$. Thus, the corresponding perceptron has one input, weight equal to -1, shifting value or bias equal to 1, and threshold equal

to 1. Observe the determinant role of the bias.

The corresponding NOT-NEURON may have a threshold equal to 0.5. So, if the input x is $x < 0.5$, which represents a zero, $1 - x \geq 0.5$ and the perceptron responds TRUE. But if the input $x >= 0.5$, which represents 1, $1 - x < 0.5$ and the perceptron responds FALSE.

31, page 22. To implement the spec function `do this but not that` with a perceptron, we can try with the following values for the parameters of the perceptron: it must have two inputs, one for p with weight 1, another for q with weight -1, $bias = 1$, and a threshold of 1.5. The sum is $S = p - q + 1$. The perceptron will output 1 or TRUE when $S \geq 1.5$. This happens if and only if $p - q + 1 \geq 1.5$ or $p > 1.5 + q - 1$ or $p > q + 0.5$. Assuming q and p are positive and less than 1, q must be less than 0.5, representing FALSE, do not do that, and $p > 0.5$, representing TRUE, do this.

37, page 26. This is a slight variation of the previous program. The code is contained in the program `PerceptronZero.html`.

38, page 26. The two perceptrons receive the same input but produce different outputs which must be kept apart. The resultant architecture is shown in the figure 5.

41, page 27. Program `allBitmaps2.html` reports bitmaps in visual mode. The Author identifies himself with the presented classification offered by the BbB distance for distances 0, 1, and 2. For distance 3, doubts arise which become stronger for distances 4 and 5. In case of strong doubt, it would be preferable to classify the bitmap as unknown.

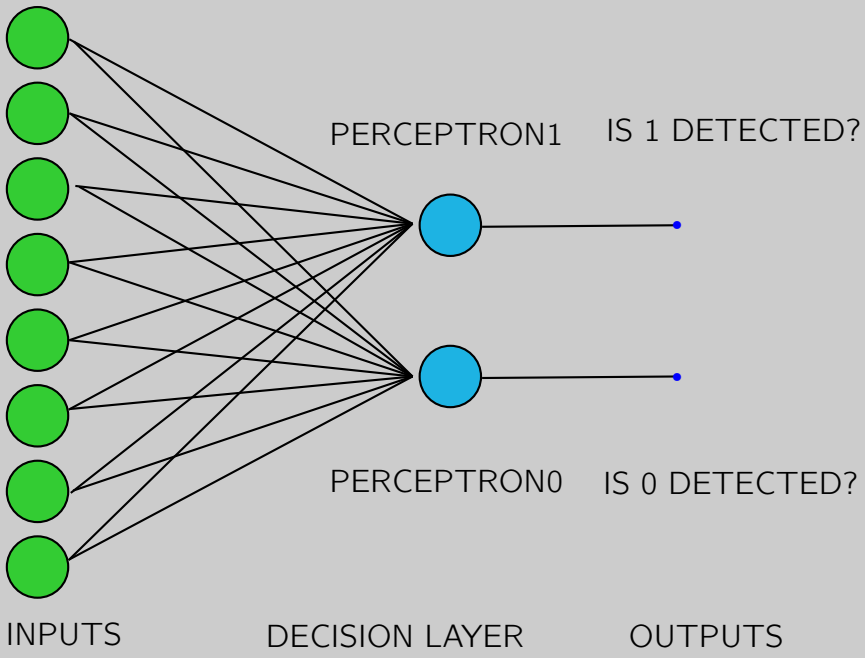


Figure 5: This NN was designed by human intelligence. Its task is to recognize slightly distorted images of 1 else 0. How does it work? It has 9 inputs, one for each pixel in the 3×3-binary-grid. The inputs represent a possibly distorted bitmap of the images of 1 else 0. All inputs are connected to each cell or perceptron of the decision layer, which contains just 2 cells. The upper cell must output 1 when an image of 1 is presented and zero otherwise. The lower cell must output 1 when 0 is presented and 0 otherwise. Images can be a bit distorted by noise and/or variability. The overall output is a string with the output of Perceptron 1 followed by that of Perceptron 0.

42, page 28. If one moderately augments the tolerance of each perceptron, everything is fine. Otherwise, errors and contradictions appear.

48, page 35. In the program `CompareTwoClassifications3.html` we gauge the performance of each pair of biases which determine the tolerance of the perceptrons. We report the number of matches, mismatches, and a score that is defined here as :

$$score = NumberOfMatches - NumberOfMismatches$$

A match happens when a perceptron agrees with the BBB distance and the other reports unknown. When both perceptrons report unknown, there is neither a match nor a mismatch. A mismatch happens in any other case and denotes a contradiction between a perceptron and the BBB distance. Our statistics read:

```
Bias of perceptron P1 = 7.8
Bias of perceptron P0 = 2.8
Score = 20
numberOfMatches 20
numberOfMismatches 0
numberOfUnknowns 492
Total 512
```

```
Bias of perceptron P1 = 8.8
Bias of perceptron P0 = 3.8
Score = 92
numberOfMatches 92
numberOfMismatches 0
```

```
numberOfUnknowns 420
Total 512



Bias of perceptron P1 = 9.8
Bias of perceptron P0 = 4.8
Score = 260
numberOfMatches 260
numberOfMismatches 0
numberOfUnknowns 252
Total 512



Bias of perceptron P1 = 10.8
Bias of perceptron P0 = 5.8
Score = 302
numberOfMatches 372
numberOfMismatches 70
numberOfUnknowns 70
Total 512



Bias of perceptron P1 = 11.8
Bias of perceptron P0 = 6.8
Score = 8
numberOfMatches 260
numberOfMismatches 252
numberOfUnknowns 0
Total 512
```

```
Bias of perceptron P1 = 12.8
Bias of perceptron P0 = 7.8
Score = -328
numberOfMatches 92
numberOfMismatches 420
numberOfUnknowns 0
Total 512
```

One sees that the tolerance can be expanded without getting contradictions until the values of the bias of $P1 = 9.8$ and $P0 = 4.8$ when 260 matches are found among 512 possibilities. Further expansion creates contradictions although the score increases. Next, contradictions overcome matches.

56, page 42. Program `EvolNN2.html` shows that our expectations are fulfilled: mismatches do not happen when 260 matches are found, which is the highest number of clean matches. The population had 20 individuals, the mutation rate was 0.2 and the recombination rate was 0.2. We raised the mutation rate because, looking at the results published to the console, we discovered that there was a stagnation point created by the discontinuity in the encoding of numbers:

The program must find the correct values of two biases, one circa 9 and the other circa 4. So, when the program offered 10 and 4, 10 got unmodifiable because you should had changed two values at the same time, putting 0 in the first digit and 9 in the second. Since the mutation rate was too low, this happened rarely, retarding the flux of Evolution. That is why we raised the values a bit to get those that they have at present, a mutation rate equal to 0.2 and a recombination rate equal to 0.2 for a population of 20 individuals. With this, the program completed

the task of getting 260 matches and no mismatch in a varying number of generations: 4, 11, 10, 11, 47, 13, 10.

57, page 42. Program `RandomBiases.html` generates the biases of an NN with values at random. It is given the task of classifying 512 bitmaps and the number of matchings of the BbB distance is reported. The whole experiment is repeated some times and an absolute frequency distribution is reported. Our final result reads: **With a relatively high frequency, randomness can produce NNs that have 256 matches and 256 mismatches, or 130 matches and 382 mismatches**.

59, page 43. Program `EvolNN3.html` uses the power of Evolution to fine-tune the threshold values of two perceptrons of the usual NN. It was run with 20 individuals, the mutation rate was 0.4, and the recombination rate was 0.4. The task was completed at generations 28, 6, 88, and 190. With a mutation rate equal to 0.3 and recombination 0.5 the numbers of generations to complete the task were: 13, 26, 57, 11, 327, 360, 61, 46, 302, 392, 3, 226.

61, page 43. Program `EvolNN4.html` shows that randomness can propose values for the weight values that produce NNs that make matches and mismatches all alike. From this, Evolution battles to diminish mismatches and then increase the number of matches. Rather soon, Evolution can reach even 232 points of 260 and without mismatches but just to fall to wander in a sea of under mediocrity that is interrupted by excursions far beyond mediocrity but not until perfection. This is independent of the values of mutation and recombination rate.

Such behavior was observed, for instance, in an experiment with a mutation rate equal to 0.1 and a recombination rate equal to 0.1, a population size of 100. More exactly, the program ran

during 100 generations and the maximal scores by generation were: 64 48 40, 40, 48 56, 56, 48 116 56 116, 116, 116, 96, 96, 96, 96, 96, 96, 96, 96, 96, -40 56, 56 64 56 74 64 116, 116 74 116, 116, 116, 116, 116, 116 32 8, 8, 64 56 96 56, 56, 56 116 96 116, 116, 116, 116, 116, 96, 96, 96, 96, 117 118, 118 116 132 16 56 116, 116 130 116, 116, 116, 118 116 132 116, 116, 116, 116 48 64, 64 48 64 116 56 96 104 112 152 116 128 116, 116, 116, 116, 96, 96, 96, 96, 56. Let us remember that the maximal attainable score is 260.

Perfection was not reached even when, in another experiment, 100000 generations were run with a population of 20 genomes, a mutation rate equal to 0.15 and a recombination rate of 0.35.

Our negative results do not allow us to claim that Evolution is false. The reason is that our failure reveals how limited is our imagination more than the incapability of Evolution itself.

62, page 44. We have tested Evolution during 100000 generations and perfection was not found. But, nobody can exclude that by generation 100001 the task is not solved.

63, page 44. We have tested Evolution over 100000 generations and we have found that Evolution wanders in a sea of under-mediocrity without giving any signal of going to an exit. So, we took this as the unmistakably signature of Evolution for the Evolution of NNs with one layer and two perceptrons. As we see, our observational window has been just to narrow to draw a more serious conclusion.

6 THE BIBLIOGRAPHY

References

[1] 3Blue1Brown 2018
But what is a neural network? . | Chapter 1 of four, Deep learning, Youtube Neural Network Tutorial in four parts.
https://www.youtube.com/watch?v=aircAruvnKk
37

[2] Mayank Banoula Last updated on May 10, 2023
What is Perceptron: A Beginners Guide for Perceptron .
https://www.simplilearn.com/tutorials/deep-learning-t
22

[3] Fodstad H.
The neuron theory. Stereotact Funct Neurosurg. 2001;77(1-4):20-4. doi: 10.1159/000064596. PMID: 12378051. .
https://pubmed.ncbi.nlm.nih.gov/12378051/
4

[4] Subrata Ghosh, Pushpendra Singh, Jhimli Manna, Komal Saxena, Pathik Sahoo, Soami Daya Krishnanda, Kanad Ray, Jonathan P. Hill & Anirban Bandyopadhyay (2022)
The century-old picture of a nerve spike is wrong: filaments fire, before membrane, . Communicative & Integrative Biology, 15:1, 115-120, DOI: 10.1080/19420889.2022.2071101
https://www.tandfonline.com/doi/full/10.1080/19420889
5

[5] Mitch Glickstein (2000)
Golgi and Cajal: The neuron doctrine and the 100th

anniversary of the 1906 Nobel Prize .
https://www.cell.com/current-biology/pdf/S0960-9822(0
23

[6] Kenneth O. Stanley , Jeff Clune, Joel Lehman, and Risto Miikkulaine
Designing neural networks through neuroevolution . Nature Machine intelligence | VOL 1 | JANUARY 2019 | 24–35 |
https://www.nature.com/articles/s42256-018-0006-z
38

[7] Diana Kwon (2023)
The quest to map the mouse brain. Nature | Vol 620 | 17 August 2023 | 686-687
https://www.nature.com/articles/d41586-023-02559-9
23

[8] Michael Nielsen (2018)
Neural Networks and Deep Learning .Determination Press 2015.
http://neuralnetworksanddeeplearning.com/
Code samples for "Neural Networks and Deep Learning"
https://github.com/mnielsen/neural-networks-and-deep-
37

[9] Piccolino M. 1998
Animal electricity and the birth of electrophysiology: the legacy of Luigi Galvani. . Brain Res Bull. 1998 Jul 15;46(5):381-407. doi: 10.1016/s0361-9230(98)00026-4. PMID: 9739001.
https://pubmed.ncbi.nlm.nih.gov/9739001/
4

[10] Jan Evangelista Purkyně. (2023, August 6)
In Wikipedia. .

https://en.wikipedia.org/wiki/Jan_Evangelista_Purkyn%
4

[11] Radcliffe, N.J. 1993
Genetic set recombination and its application to neural network topology optimisation. Neural Comput & Applic 1, 67–90
https://link.springer.com/article/10.1007/BF01411376
38

[12] Rodríguez José 2023
Christianity and the Avatar Interpretation of the Brain v2 .
https://www.ejristos.com/eternity/files/AvatarBrain.p
17

[13] F. Rosenblatt, 1960
Perceptron Simulation Experiments . in Proceedings of the IRE, vol. 48, no. 3, pp. 301-309, March 1960, doi: 10.1109/JRPROC.1960.287598.
https://ieeexplore.ieee.org/abstract/document/4066017
16

[14] tutorialspoint. Undated. Active by 2023.
Logic Gates .
https://www.tutorialspoint.com/computer_logical_organ
9

[15] W3schools (2023)
Perceptrons .
https://www.w3schools.com/ai/ai_perceptrons.asp
20

[16] Xin He (University at Buffalo) undated, active by 2023
Propositional Logic .
https://cse.buffalo.edu/~xinhe/cse191/Classnotes/note
9

[17] Xin Yao 1993
A Review of Evolutionary Artificial Neural Networks .
International Journal of Intelligent Systems Vol 8: 539-567
https://onlinelibrary.wiley.com/doi/epdf/10.1002/int.
38

[18] Yoder, Lane. (2009).
Explicit Logic Circuits Discriminate Neural States. PloS
one. 4. e4154. 10.1371/journal.pone.0004154.
https://journals.plos.org/plosone/article?id=10.1371/
22

All cited links were active by 8/XI/2023