EVOLUTIONARY NEURAL NETWORKS
3. DIVIDE AND CONQUER

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

November 15, 2024

Abstract

We investigate the possibilities of evolution in building neural networks (NNs) that recognize images. In this article, we explore the effect of dividing in small portions the grid where drawings appear.

Contents

1 INTRODUCTION 3

2 THE OLD ARCHITECTURE 5

3 THE NEW ARCHITECTURE 9

4 TOLERANT NNs 14

5 TOLERANT EVOLUTIONARY NNs 15

6 CONCLUSION 21

7 ANSWERS TO EXERCISES 21

8 THE BIBLIOGRAPHY 31

1 INTRODUCTION

Our aim is to use Evolution to design Neural Networks (NNs) that classify images. We work with toy models to get insight and to allow calculations to be done in ordinary computers. Our images consist of patterns that are drawn in a 3×3 grid whose pixels are white else black and that therefore can be described by binary strings 9 bits long. These are called bitmaps.

Each bitmap is classified by the criterion of the closest neighbor according to the BBB-distance as a deformation of one of two archetypes, an image of 1 and another of 0. The BBB distance is the Bit-By-Bit distance which counts the number of places where two binary strings differ.

We already have designed an NN with two perceptrons that can correctly classify 260 bitmaps out of 512. Our perceptrons are indeed generalized boolean gates that are tolerant to noise and variability. We proved in our *Intro* that Evolution can adjust separate sets of diverse parameters of the NN to tune it. (Rodriguez, [2] 2024). We also proved that perfection, to correctly classify all 512 bitmaps, is attainable but up to an exaggerated cost of nearly 250 perceptrons. We also showed that Evolution has some power to simplify the NN (Rodriguez, [3] 2024b).

This time, we get inspiration from the old strategy in Computing Science of *Divide and Conquer*, which divides a complex problem in sub-problems, whose solutions shall be combined to generate a solution to the original problem (geeksforgeeks, [1] 2024). Thus, we divide the 3×3 grid in 3 rows. This causes that each bitmap is divided in three sub-bitmaps that must be recognized by separated circuits, all of which shall be joined to produce a final classification of whole bitmaps.

The archetypes look like this, that on the left represents number 1, that on the right, 0.

```
010     111
010     101
010     111
```

The corresponding bitmaps are 010010010 and 111101111.

Now, we have the same grid, but each archetype, as well as each image, have three associated bitmaps. Those of the archetype 1 are 010 010 010 and those of the archetype 0 are 111 101 111. This is our great invention. The same strategy has been used to pass from words to syllables, that allowed ancient Egyptian to invent a writing system, and from words to phonemes, that allowed Phoenicians to invent the alphabet. The net result is a great gain of expressibility and fine-tuning. Thus, we hope that our adopted scheme would allow deciding the parameters of the recognizing perceptrons locally and therefore the stability against noise and variability would be enhanced.

Our first problem now reads: devise with our own mind an NN that takes as input the 3 sub-bitmaps of each image and reproduces all results of the NN with the old architecture with two perceptrons. The second problem is to have Evolution to chase the parameters of the NN to fit the same results.

Our plan is to reuse what we know to get further. So, a review is convenient at this moment.

2 THE OLD ARCHITECTURE

We joined two perceptrons in a Neural Network (NN), see figure 1, and showed that it can correctly classify 260 of the 512 possible bitmaps where the BBB distance defined the correct assignation. Each bitmap was assigned to the closest archetype according to this distance. Some details follow:

A perceptron must decide if a stimulus is strong enough to fire it. The force of the stimulus is given by a total sum S, and it must be stronger than a certain threshold T. So, we must decide if $S \geq T$.

Let us proceed step by step. Our task is to design a perceptron that fires when it receives a specific stimulus. The stimulus is a bitmap, in this case 010010010, the bitmap of the archetype 1.

The first step is to associate a boolean clause to the bitmap of the archetype. Boolean means that it refers to TRUE else FALSE. A variable can be negated, an operation that inverts the output. A clause can have various variables that are joined by suitable operators, AND, OR.

The boolean clause associated to a bitmap is constructed as follows: a 1 in the bitmap is replaced by a boolean variable, and a 0 by a negation of a boolean variable. Concretely:

The bitmap of 1, 010010010, induces the boolean clause

$$\sim n_0 \wedge n_1 \wedge \sim n_2 \wedge \sim n_3 \wedge n_4 \wedge \sim n_5 \wedge \sim n_6 \wedge n_7 \wedge \sim n_8$$

The $\wedge$ is the AND-boolean operator that says that a bitmap is really a list of conditions that must be all fulfilled to detect a pattern. The subindexes keep track of the position of each bit.

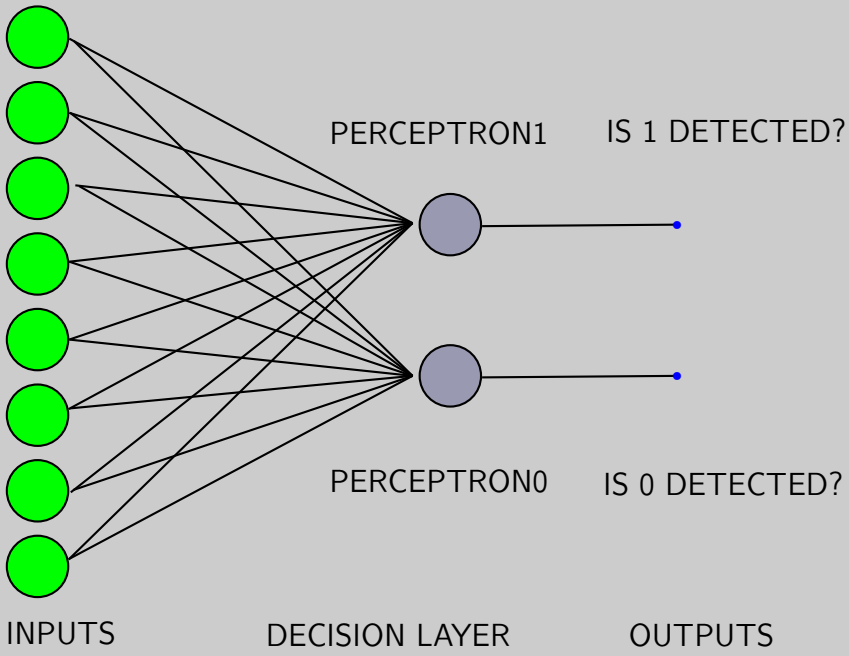


Figure 1: This NN was designed by human intelligence. Its task is to recognize distorted images of 1 else 0. How does it work? It has 9 inputs, one for each pixel in the 3×3-binary-grid. The inputs represent a possibly distorted bitmap of the images of 1 else 0. All inputs are connected to each cell or perceptron of the decision layer, which contains just 2 cells or perceptrons. 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. Warning: this output must be analyzed to see whether it contains a contradiction. This happens when both perceptrons output 1, that means that the first perceptron has perceived a 1 and the other a 0. If the output is 10, it says that the first perceptron perceived a 1, while the other found nothing. The NN perceived a 1. If the output is 01, the input was unknown for the first perceptron, while the second perceived a 0. The NN perceived a 0. There is here no contradiction. The same happens for the output 00, which is interpreted as UNKNOWN. Thus, the logic of this NN is three-valued: 1 has been detected, 0 has been detected, UNKNOWN.

A clause is a function: for each string of length 9 of boolean values, it outputs either TRUE else FALSE. It is the **spec function**. If the output is TRUE, we say that the given bitmap has been recognized as the archetype. If the output is FALSE, we get in trouble: the perceptrons that will implement our ideas, fires else not. When it fires, if it responds TRUE, the archetype has been recognized. But if it does not fire, we must not invent an answer: we just do not know what happens. The output is UNKNOWN.

The next step is to associate an algebraic expression in real variables to the boolean clause. So, we associate an affine dichotomous expression to each boolean clause. Dichotomous means that each variable can take only on values 1 else 0. Affine means that a constant could be added.

We have two replacement rules that associates a first degree real polynomial to each char of the bitmap or variable of the associated clause:

- $1 \to e_i$,
- $0 \to -e_j + 1$.

where each variable is marked by a subindex that declares its position in the bitmap beginning from index 0. The coefficient of the variable is the weight, the accompanying constant is the bias, the sum of all biases is the total bias. For an AND-perceptron the threshold T is the number of bits because to answer YES or ONE, all entries must be activated.

Thus, in the old architecture, the sum S associated to the perceptron that fires after the registration of the archetype 1, with bitmap 010010010, is:

$$S = -e_0 + 1 + e_1 - e_2 + 1 - e_3 + 1 + e_4 - e_5 + 1 - e_6 + 1 + e_7 - e_8 + 1 = -e_0 + e_1 - e_2 - e_3 + e_4 - e_5 - e_6 + e_7 - e_8 + 6$$

where we counted bits from 0 to 8. Since there are 6 zeroes, the bias is 6. This sum S must be greater, stronger, than the threshold, T, to fire the perceptron: $S \geq T$. The value of T for an AND-perceptron is the length of each bitmap, 9. At last, the inequality $S \geq T$ reads:

$$S = -e_0 + 1 + e_1 - e_2 + 1 - e_3 + 1 + e_4 - e_5 + 1 - e_6 + 1 + e_7 - e_8 + 1 = -e_0 + e_1 - e_2 - e_3 + e_4 - e_5 - e_6 + e_7 - e_8 + 6 \geq 9$$

If one replaces the variables by the corresponding bits, we get $-0+1-0-0+1-0-0+1-0+6 = 9 \geq 9$. Since this is true, we say that the archetype1 has been recognized by the perceptron.

Let us notice that the NN recognizes no other bitmap. Because, other bitmap will be different from that of the archetype1. If a 0 is changed by a 1, the left side of the inequality receives -1, and if the bitmap has a 0 instead of a 1, the left hand of the inequality loses a 1. Thus, any discrepancy with the archetype1 causes a lost of the strength of the stimulus: the perceptron will not fire anyhow.

1 *Exercise.* *Write down the inequality $S \geq T$ corresponding to the archetype0, 111101111. How many bitmaps are recognized by the corresponding perceptron?* Answer

The two aforementioned inequalities represent the architecture of figure 1 with two perceptrons, one that recognizes the archetype1, representing 1, and the other, archetype0, representing 0.

3 THE NEW ARCHITECTURE

We divide the grid in 3 rows, and devise an analyzing system for each one of them, whose outputs must be somehow joined to produce a unique response.

Our task is to pass from a system, whose input is a description of a bitmap as a single string, such as 010010010, to another in which bitmaps are described by 3 triads, as 010 010 010. Each description generates its own architecture. The first is the old, the second is the new that must be designed.

To fill in the details of the new architecture, let the perceptron description guide us. The inequality $S \geq T$ of the old architecture is best suited for us in its primitive form:

$$S = -e_0 + 1 + e_1 - e_2 + 1 - e_3 + 1 + e_4 - e_5 + 1 - e_6 + 1 + e_7 - e_8 + 1 \geq 9$$

Let us remember: when the bit is 1, the weight is 1 and the bias is 0, if the bit is 0, the weight is -1 and the bias 1, and we deal here with the archetype 010010010. This inequality can be rewritten as

$$S = (-e_0 + 1 + e_1 - e_2 + 1) + (-e_3 + 1 + e_4 - e_5 + 1) + (-e_6 + 1 + e_7 - e_8 + 1) \geq 3 + 3 + 3$$

where we have applied the associative law of real numbers to inputs grouped by triads. We immediately see here 3 perceptrons that recognize each one a third of the original bitmap:

$S_0 = (-e_0 + 1 + e_1 - e_2 + 1) \geq 3$, perceptron P_{10}
$S_1 = (-e_3 + 1 + e_4 - e_5 + 1) \geq 3$, perceptron P_{11}
$S_2 = (-e_6 + 1 + e_7 - e_8 + 1) \geq 3$, perceptron P_{12}

where the right 3 represents the fact that in the new architecture, we deal with bitmaps of length equal to 3. Each inequality represents a perceptron that recognizes the sub-bitmap 010. What we need now is to join them together to recognize the bitmap of the archetype 010010010. That task is done by perceptron P_1 that we specify now:

Is perceptron P_1 AND, OR, NOR, or NAND? Our answer is: the decision perceptron P_1 must be of type AND because to recognize the bitmap of an archetype, all three sub-bitmaps must be recognized at the same time.

We have defined in that way the architecture presented in figure 2 and from another perspective in figure 3. The inputs are connected to sensory cells P_{ij} that conform the so-called hidden layer, whose output feeds the decision layer that contents the cells that must recognize the patterns, P_1 for the archetype of ONE and P_0 for ZERO. The cells that feed P_1 are P_{10}, P_{11} and P_{12}. Those that feed P_0 are P_{00}, P_{01} and P_{02}.

We have implemented these ideas in program `divideP1.html`. The code is the following:

```
<!DOCTYPE html>
<html>

<body>
    <h1><h1>Program divideP1.html
        <br> Perceptron P1 and its slaves</h1>
    <p>The bitmap of archetype 1, 010010010, must be recognized
        <br>by an NN that divides the bitmap in three parts,
        <br>010 010 010, which are processed by three perceptrons
        <br>of the hidden layer,
        <br>whose output feed the decision perceptron P1.
    </p>

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

    <script>
        "use strict";

        const archetype = "010010010";

        //Output of generic perceptron
        function outputPerceptron(inputs, weights, bias, threshold, name) {
            let sum = 0;
            let output = 0;
            for (let i = 0; i < inputs.length; i++) {
                sum += inputs[i] * weights[i];
            }
            document.getElementById("myDoc").innerHTML +=
                "<br>" + name + " :  sum = " + sum + ", bias = " + bias;
            sum += bias;
            if (sum >= threshold) { output = 1 }
            return output;
```

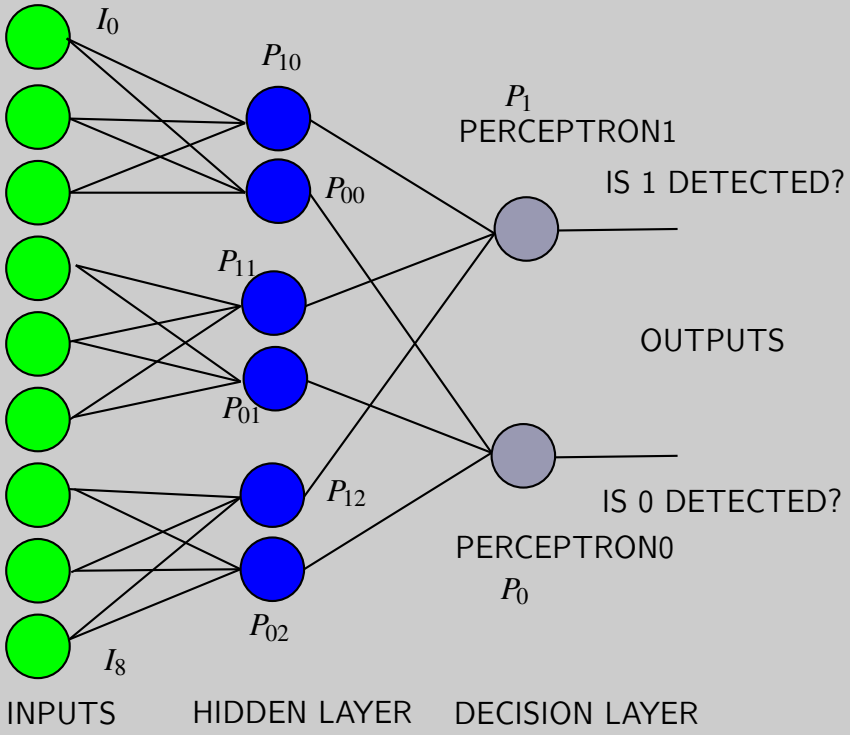


Figure 2: This NN is our first implementation of our directive *divide and conquer*. Its task is to recognize distorted images of 1 else 0. How does it work? It has 9 inputs, one for each pixel in the 3×3-binary-grid. The inputs, I_0 ... I_8, represent a possibly distorted bitmap of the images of 1 else 0. The output of the NN is produced by perceptrons $P1$ and $P0$. The first fires when perceives an image that represents 1, while the second fires when perceives a 0. The Inputs are grouped by triads following the reading order of the pixels in the grid, which they represent. Each triad of inputs is analyzed by two perceptrons of the hidden layer, the first, P_{1j} works for P_1, and the second, P_{0j} for P_0. Our task is to define the parameters of each perceptron to exactly reproduce the behavior of the old architecture that have only one pair of perceptrons and does not group the inputs anyhow.

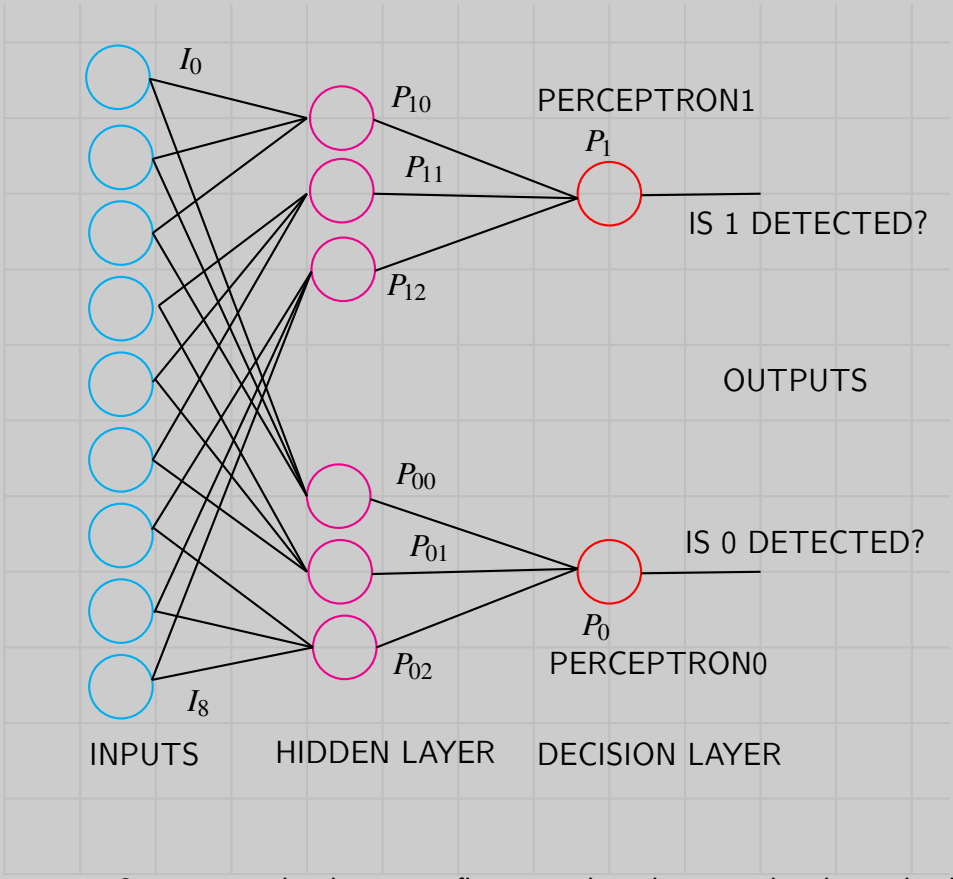


Figure 3: One can easily observe in figure 2 that the input has been divided in three parts. The architecture of the NN is not visible at once. But in this new figure, that has been resolved. It has been made more clear, more transparent, more modifiable, more evolvable. When things get complex, small giros in insights and corresponding presentations can make a huge difference in evolvability, the capability to be easily complexioned, including more function or expanding its domain.

```
        }

        function main() {
            //All three AND-perceptrons of the hidden layer
            //have the same weights, bias and threshold
            //because all are target to the same sub-bitmap.
            let weights = [-1, 1, -1];
            let bias = 2;
            let threshold = 3;
            //The bitmap of the archetype
            //is divided in three parts
            let bm0 = archetype.substring(0, 3), //010
                bm1 = archetype.substring(3, 6), //010
                bm2 = archetype.substring(6);  //010
            document.getElementById("myDoc").innerHTML +=
                "Archetype = 010010010.  <br> Subparts :  "
                + bm0 + "  " + bm1 + "  " + bm2;
            //HIDDEN LAYER
            //Perceptron P10
            let inputs = bm0;
            let name = "P10";
            let output0 = outputPerceptron(bm0, weights, bias, threshold, name);
            //Perceptron P11
            inputs = bm1;
            name = "P11";
            let output1 = outputPerceptron(inputs, weights, bias, threshold, name);
            //Perceptron P12
            inputs = bm2;
            name = "P12";
            let output2 = outputPerceptron(inputs, weights, bias, threshold, name);

            //DECISION LAYER
            //processes  the outputs of the hidden layer
            //with an AND-perceptron assuming that
            //each perceptron of the hidden layer
            //recognizes its sub-bitmap.
            //So, their outputs must be all 1.
            inputs = [output0, output1, output2];
            let weightsP1 = [1, 1, 1];
            bias = 0;
            threshold = 3;
            name = "P1";
            let outputP1 = outputPerceptron(inputs, weightsP1, bias, threshold, name);
            let finalOutput = "NO";
            if (outputP1) { finalOutput = "YES" };
            document.getElementById("myDoc").innerHTML +=
                "<br><br>Does this NN recognize the archetype?   <br>" + finalOutput;

        }

        main();


    </script>

</body>

</html>
```

2 *Exercise.* *Run program* `divideP1.html` *and verify that it recognizes the archetype 1.*

3 *Exercise.* *How many bitmaps does the NN of program* `divideP1.html` *recognize? Compose a verbal proof plus a program that tests it.* *Answer*

4 *Exercise.* *Design an NN that recognizes the archetype 0 that has been divided in three parts. You must specify the architecture and the parameters of all its perceptrons.* *Answer*

We have two circuits of perceptrons, the first recognizes the archetype 1, the second the archetype 0. Our next task is to join them together to get a functional NN. So, we must develop the code of an NN that accepts only the archetypes of 1 and 0 and correctly classify them. If the input is the archetype of 1, the NN must output 10, if it is that of 0, the output must be 01. If the input is any other, the output must be 00. The output 11 must never appear because that amounts to a contradiction: it means that a given bitmap represents both archetypes, that of 1 and that of 0. Our solution to this task appears in program `divideConquerNN.html`

5 ***Exercise.*** *Run program* `divideConquerNN.html` *and verify that it recognizes the two archetypes, of 1 and 0, and nothing else. To archetype1 it must respond* 10*, to archetype0* 01*, and to any other input* 00*. The output* 11 *never must appear. Answer*

6 ***Challenge.*** *Program* `divideConquerNN.html` *correctly works, but its code is redundant because the two methods that recognize the bitmaps of the two archetypes are almost equal. Simplify the code.*

We have now an NN that recognizes two triads of sub-bitmaps, the first is of the archetype 1, and the second of that of 0. All perceptrons in this NN are of type AND. We face now to another task: to implement tolerance to noise and variability.

4 TOLERANT NNs

We have designed an NN with two circuits of perceptrons that recognizes only two archetypes. Each circuit recognizes a specific archetype. That happens because we have been dealing with an equation: $S = T$ that says that the stimulus must be exactly equal to the threshold. If some mismatches happen, $S < T$, the stimulus loses strength and the perceptron cannot fire. Thus, to make our NN failure tolerant,

we add a term, $addBias$, the additional bias, to compensate the deficit created by mismatches. In that way, we get $S + addBias > T$. The problem is the possible appearing of contradictions that happen when both circuits are fired, when they both proclaim that the input bitmap is a recognizable deformation of its assigned archetype.

So, the problem is to find the values of the additional biases of all 8 perceptrons to maximize the number of firings without generating contradictions.

7 Challenge. *Develop a program to explore the possible values of the additional bias of each one of the 8 perceptrons of our RN. Make a prognosis of the maximal number of recognized bitmaps, but without falling into contradictions either between circuits and against the BBB distance.*

Let us pose to Evolution the task of optimizing the values of the additional biases to maximize matches but without contradictions both between circuits and against the BBB distance.

5 TOLERANT EVOLUTIONARY NNs

The Evolutionary Process must specify the values of the additional biases of all 8 perceptrons of our NN, which until here have been 0. This family of parameters is responsible for tolerance to noise and variability, i.e. it enables the recognition of distorted images as deformations of one of the two given archetypes.

Since the only task posed to Evolution is to specify the 8 values of the additional biases, our chromosomes must encode their values and nothing else.

How will we encode the additional biases? We used in a previous program 4 chars for each bias, say the string 1234 encoded for the real number 12.34, the value of one bias. Now, let us reason like this: the BBB distance between two bitmaps takes on integer values, 0, 1,...,

9. Therefore, the additional biases must take just on the same values to encircle more and more bitmaps beginning from each archetype. Hence, our chromosomes must have one char for each bias, and we have 8 perceptrons, therefore, we need 8 chars per chromosome. Let us list the perceptrons: P10, P11, P12, P1 , P00, P01, P02, P0. A chromosome would look like this: 20628543 and the list establishes the correspondence with the biases of the perceptrons.

To begin with, let us modify the code of program `divideConquerNN.html` to be compatible with Evolution and with our purpose: the NN must enlarge its tolerance and therefore the program must be capable of reading values for the additional biases of all its 8 perceptrons. In the case of Evolution, the code must correctly incorporate the additional biases which must be read from the chromosomes. Keep in mind that these are strings, while the biases are numbers. We must register the effect of changing the values of additional biases on the number of accepted bitmaps by the new NN but without contradictions.

Our first step to solve the task is in program `divideConquerNN2.html`. There, a string 8 chars long was defined that contains the 8 values of the additional biases. This string was next transformed to a vector of integer numbers and its values applied to the 8 perceptrons in the order P10, P11, P12, P1 , P00, P01, P02, P0. The first 4 were P1 and its slaves, the other 4 were P0 and its slaves.

8 *Exercise.* *Run program `divideConquerNN2.html` with all 8 values of additional biases equal to 0 and verify that this NN accepts only two bitmaps, the two archetypes.* *Answer*

Next, we devised program `divideConquerNN3.html` to generate random values of the additional biases and to test them on random bitmaps.

9 *Exercise.* *Run program `divideConquerNN3.html` and draw your conclusion about the appearing of contradictions.* *Answer*

Finally, we have in the program `divideConquerEvolNN.html` our implementation of the Evolutionary Optimization of the additional biases.

10 ***Exercise.*** *Satisfy your curiosity playing with program* `divideConquerEvolN` *to watch how Evolution finds the maximal values for the additional biases of the 8 perceptrons of the given NN but without falling into contradictions.* *Answer*

11 ***Mystery.*** *As reported in our Intro, we must classify 512 bitmaps as deformations of 1 else 0 according to the BBB distance but without contradictions. Our initial RN with 2 perceptrons that accepted as input complete bitmaps was able of recognizing 260. Here we have experimented with the idea of dividing a large grid into subgrids. Our hope was that in the same way as our Phoenician alphabet allows more expressibility and finely tuned recognizing of messages, subpatterns associated to subgrids also could improve results in the task of visual pattern recognition with bitmaps. But we got a rampant failure, which is for us a mystery. Because we expected from our results to be better than the former ones, but instead the performance was divided by two.*

We failed, and, up to this line, we do not understand why.

After thinking for some time, we got some ideas to explain our mysterious failure.

The first possible cause is that the encoding of Evolution might be defective. But we consider that Evolution works here because if mutation and recombination are moderate to low, we see that the top fitness increases. Such as expected. But if these are high, Evolution wanders around without rest. Such as expected. So, we do not consider that the code could be wrong.

What else?

We expected to improve the performance of an old NN with two perceptrons. But our results say that performance was divided by two.

Seemingly, it is not the code. Thus, let us consider the possibility that **we have been working with not explicitly expressed wrong concepts**. So, let revise our work and ideas. In a nutshell, all we have been doing is to compare two systems of inequalities, the first corresponding to perceptrons of the old architecture, the second dealing with a system of inequalities in which a bitmap is divided in three parts with separated recognizing machinery.

The perceptron of the old architecture that recognizes the deformations of the archetype 1 is described by the inequality:

$$S = (-e_0 + 1 + e_1 - e_2 + 1) + (-e_3 + 1 + e_4 - e_5 + 1) + (-e_6 + 1 + e_7 - e_8 + 1) + addBias1 \geq 9 = 3 + 3 + 3 = T$$

The corresponding part of the new architecture is depicted by the following system of inequalities

$$S_0 = (-e_0 + 1 + e_1 - e_2 + 1) + addBias10 \geq 3,$$
$$S_1 = (-e_3 + 1 + e_4 - e_5 + 1) + addBias11 \geq 3,$$
$$S_2 = (-e_6 + 1 + e_7 - e_8 + 1) + addBias12 \geq 3,$$

where the right 3 represents the fact that in the new architecture, we deal in all three cases with bitmaps of length equal to 3. We have explicitly incorporated the additional biases responsible for failure tolerance, which are positive integer values less than 9.

At its face value, we have divided one large inequality in three. Like this: from $7 = 1 + 2 + 4 > 2 + 2 + 2 = 6$, we extract three inequalities, $1 > 2$, $2 \geq 2$, $4 > 3$. We got two valid inequalities and one false. So, this reading, that fed our imagination, is misleading. But **the Scientific Method, of pursuing predictions with experiments and critical thought, has freed us**.

On the other hand, we can add three inequalities of positive real

numbers to get a sound inequality: from $3 > 2$, $2 > 1$, $3 > 1$ we can get $6 = 3+2+1 > 2+1+1 = 4$. Now, since the inequality corresponding to the perceptron of the old architecture can be deduced from the corresponding systems of inequalities of the new one, by adding them, we get a prediction:

If the new architecture fires, the old also. That is, the old is not worse than the new. This is an open contradiction with what pushed us to do this work, because we expected the new architecture to be better than the old one. Our fundamental concepts rested on an analogy with phonetic alphabets that at last proved to be wrong. We witness that an inspiring slogan of Computer Science -*divide and conquer*- does not function here. Thus, it is not unconditionally correct.

Let us continue with other questions.

12 *Exercise.* *It seems to us that the old architecture is not worse than the new and that most possible, the old is the best. Prove this. To achieve this, you must verify the following two predictions. First: if a bitmap is recognized by the new architecture, it is also recognized by the old one. Second: There are bitmaps that are recognized by the old architecture but not by the new one.* Answer

13 *Exercise.* *To add inequalities of positive real numbers and with the same direction is equivalent to congruently add weights from diverse scales. Thus, one can add inequalities of positive real numbers to produce a sound inequality. From this we concluded that the old architecture extends or generalizes the new one, in its various degenerate versions, meaning that when the new architecture fires, the old must also fire. But direct counting showed that this is false: there are 6 events, in which the new architecture fired but not the old one. Therefore,* ***no NN generalizes the other one****. But, we have been thinking in the last minutes that the old architecture generalizes the new one. Detect the cause of our confusion.* Answer

14 ***Exercise.*** *Redact the whole story of this article as an application of the Scientific Method.* *Answer*

Our main results are:

1. We have considered two NNs, the old one with two perceptrons, each one with many inputs, the new with 8 perceptrons and fewer inputs to each perceptron. In sensu stricto, the two NNs are independent, and must be studied separately.

2. Our implementation of the strategy *divide and conquer* did not work. So, it is not unconditionally correct. Maybe our approach is wrong: it is not the grid that must be divided, but the symbols drawn on the grid themselves.

3. The old architecture statistically generalizes the new one: in nearly 90% of cases, when the new architecture fires, the old also.

4. The old architecture has 2 cells while the new one has 8, and if we consider degeneration, the number of cells can surpass 32. So, the new architecture is not good for the planet. Degeneration refers to the fact that the new architecture needs at least 4 different chromosomes, specifications of tolerance, additional biases, to try to be as good as the old NN.

15 ***Application.*** *Our findings can be expressed succinctly: NNs whose perceptrons have many inputs to analyze large images are cheaper and more effective than systems of perceptrons with fewer inputs to analyzes subpatterns. This is de facto an explanation of the fact that neurons always have so many dendrites or inputs whose number can reach 1000.*

16 ***Challenge.*** *Recast our battle with complexity according to an axiomatic format, i.e., when everything correctly flows beginning from true principles.*

17 *Graduation.* *Our thesis is that neuronal networks, both biological and artificial, whose units have many inputs to analyze complex patterns are better than systems with many subsystems that analyze subpatterns. The final justification is that the former can correctly classify more bitmaps and at a lower cost. Dear Reader: make fun of the Author of this article by clearly stating his supporting evidence in favor of this thesis. Now is the right time for you to do something better.*

6 CONCLUSION

NNs are complex entities whose study can easily tangle up the researcher. In our case, we thought that we could make a great profit by applying the known strategy of Computer Science of *Divide and conquer*. But we failed. Nevertheless, we used the Scientific Method to help ourselves to understand the multiples and at last simple causes of confusion. That method compares predictions of beliefs with observed results and with ruthless discussion. We found this method to be a great myth buster. Anyway, we consider that NNs with a large number of inputs that are targeted to large patterns are cheaper and have more recognizing capabilities than conglomerates of small NNs that point to subpatterns. This is proposed as an explanation of why neurons are obsessed with large numbers of inputs or dendrites.

7 ANSWERS TO EXERCISES

1, page 8. The bitmap of archetype0 is 111101111. Therefore, the corresponding inequality reads:

$$S = e_0 + e_1 + e_2 + e_3 - e_4 + 1 + e_5 + e_6 + e_7 + e_8 =$$
$$e_0 + e_1 + e_2 + e_3 - e_4 + e_5 + e_6 + e_7 + e_8 + 1 \geq 9$$

If one replaces the variables by the corresponding bits, we get $1+1+1+1-0+1+1+1+1+1=9\geq 9$. Since this is true, we say that the archetype0 is recognized by this perceptron. No other bitmap is recognized by this perceptron because any change will be chastised with a lost of strength in the stimulus and the umbral will not be reached.

3, page 13. The NN of program `divideP1.html` recognizes only one bitmap. The reason is that each perceptron of the hidden layer recognizes one and only one bitmap. Additionally, the decision perceptron P_1 is an AND perceptron that demands all its inputs to be one or true. Therefore, the whole ensemble also recognizes only one bitmap. Our reasoning is tested by program `divideConquerP1b.html` where the NN is fed with random bitmaps. None is expected to be recognized unless it matches the archetype representing 1. The test has been passed.

4, page 13. The idea is exactly the same as in the problem of recognizing the archetype 1. Nevertheless, the three sub-bitmaps are not the same and hence, the code observes a minor variation. Concretely:

The bitmap of the archetype of 0 is 111101111. The three sub-bitmaps are 111 101 111. Hence, the parameters of the AND-perceptrons that recognize them are:

```
For 111:
inputs = [1,1,1]
weights = [1,1,1]
bias = 0
threshold = 3

For 101:
inputs = [101]
```

```
weights = [1, -1,1]
bias = 1
threshold = 3
```

The perceptrons of the hidden layer are connected to the decision perceptron P0, which is an AND-perceptron. The corresponding code is in program `divideP0.html`.

5, page 14. The program is correct.

8, page 16. The test was passed: when the additional biases are all set to zero, only the two archetypes must be recognized because this amounts to establish an equality $S = T$.

9, page 16. Explorations rapidly render that when tolerance is high, i.e., when additional random biases take on integer values, 0,1,...,9, contradictions abound. But when tolerance is low, i.e., when additional random biases take on values 0,1 and 2, contradictions may not appear.

10, page 17. We begin our study of the power of Evolution to tune the values of additional biases to maximize matches without contradictions by using low values of the additional biases. That was controlled by the *usedAlphabet* variable, line 150. For low tolerance, it is 01, for high values it is 0123456789. We begin to work with a population of 100 individuals.

To test the code, we used only 0 as a possible value for all the 8 additional biases. The output was correct: when there is no additional bias, only the archetypes are recognized, and all the other must remain as unknown.

Next, we inquired what happens when we expand the alphabet. To begin with, we use the alphabet 01: in the initial population, the number of matches reached 128 under 0 mismatches, so, the number

of unknowns were 384 and there was no further Evolution during 10 generations.

With the alphabet 012 mismatches disappeared by generation 18 with 128 matches and 384 mismatches. No further progress until generation 30, when the program halted.

Alphabet 0123: because of troubles with memory, the population size was lowered down to 30. Besides, mismatches never disappeared, so we chastised mismatches stronger. The score function used until here was:

```
score = numberOfMatches - 7 * numberOfMismatches;
```

We will use now :

```
score = numberOfMatches - 20 * numberOfMismatches;
```

The dynamics, according to the best result of each generation, got stabilized before generation 50: 128 matches, 384 unknowns and 0 mismatches. Observed during 500 generations.

With the same parameters, we run the program with the alphabet 01234: population was set to 25 to alleviate memory problems. Evolution achieved after 700 generations a maximum of 64 matches under 0 mismatches. For this reason, the score function was redefined to

```
score = numberOfMatches - 13 * numberOfMismatches;
```

The chastise for mismatches was lowered in order to allow swifter Evolution. Nevertheless, the situation remained equal. But since we felt attormented by the desire of knowing whether 128 was the maximal number of matches without mismatches, we were forced to run the program during many more generations. Thus, we gave the program an opportunity in a laptop with 16 Gigs (we have been working with 8 Gigs with Mozilla under Ubuntu Linux 24.04).

With a population of 75, alphabet 01234 , the program was run during 100000 generations. By generation 100, the achieved number of matches was 92 under 0 mismatches and remained invariant for the rest of life. We conclude that the alphabet 01234 is too noisy and perturbs the assertive role of Evolution. But to exclude any false conclusion due to hasting, we run the alphabet 0123 during 100000 generations . In this case, by generation 100 already 128 matches were reached without one single mismatch and the situation remained invariant during 9900 more generations.

What could be the result when we run the alphabet 012 during 100000 generations? 92 matches were reached under 0 mismatches by generation 100. The situation remained frozen for the rest of life. The winner chromosome was the same all along. In another run, 128 matches with the chromosome 11101110 were found in the first generation, without mismatches. This situation remained unchanged during 100000 generations. In a third run, a worthless champion was different in generation 100 from that of the first generation. No change during the following 99900 generations.

And, what about the alphabet 01? The maximal value was attained in the initial generation: 128 matches at zero mismatches with the chro 11100101. The situation was frozen during 99999 more generations.

Our main worry is: why does everything get rapidly frozen? Our first idea was that the mutation and recombination rates were too low. So, instead of a mutation rate of 0.0005 and a recombination rate of 0.1, the mutation rate was amplified 10 times and the recombination rate 3 times. The alphabet was 012. Result: the maximal value, 128 matches under 0 mismatches, was found already by the initial generation. The corresponding chromosome was 11100101 but then, the number of matches felt to 92 by generation 100 with chromosome 11101200. The value 128 was recovered with the chromosome 11101110 by generation 3800 and remained so until generation 100000. All that happened while the number of mismatches always

was 0.

The net conclusion is that the maximal number of matches of this RN with 8 perceptrons is 128, which is almost the half of the number that was attained by the ancient RN with 2 perceptrons that was considered in our *Intro*. This is a great failure for us, and we have no idea of the cause, apart from imagining that the achieved variability was not representative. Thus, an experiment is due: to warm the variability by augmenting the mutation and recombination rates while diminishing the population size in order to increase sampling effects. So, larger regions of the available space might be visited and the 128 gets challenged. Thus, the mutation rate was set to 0.05 and the recombination rate to 0.4, with a population size of 25 (instead of 75) and programs were run during 200000 generations.

Results: Diverse values of matches were reported as the best of the generation, with and without mismatches. The highest value without mismatches was 128. Thus, we firmly believe that 128 is the maximal value that the new NN can attain.

Each one of these experiments takes nearly 80 minutes over a dedicated old laptop under Windows 10 and Microsoft Edge with 16 GB of ram.

12, page 19. While the question seems to admit a simple answer, we must clarify a subtlety before returning to it.

Comparison of algebraic inequalities have allowed us to make seemingly correct deductions. So, let us continue with them, but this time with critical thinking. Passing from the old architecture to the new one, implies the division of the total additional bias in the old architecture in 3 parts in the new one. This dividing procedure can be done variously. For instance, if the total additional bias is 2, we can divide it in three parts in various forms: as $0+1+1$, $1+0+1$, or $1+1+0$, we say that the division is degenerate. But, attention, the Evolutionary Process finds an optimal chromosome which is constant, it can not be varied unless we run the program one more time.

From here, we get a prediction:

The maximal value, 128 matches with 0 mismatches, is degenerate, i.e., it can be attained with various chromosomes that must be found in diverse runs.

To test the prediction, we run variously program `divideConquerEvolNN.html` with a population size of 100 during 10 generations.

We got the following chromosomes encoding for 8 additional biases that produced the maximal number of matches, 128:

```
01010101
01011110
11100101
11101110.
```

The prediction was verified: the optimal additional bias of the old architecture corresponds not with one chromosome but with a group of them in the new architecture. So, we do not exclude the possibility that the complete set of optimal chromosomes of the new architecture could be as good as the old architecture or even better. This must be verified and to that aim, we developed program `divideConquerEvolNN2.html` that takes a chromosome and generates the corresponding reclassification of all the 512 bitmaps, 256 of which were assigned by the BBB distance as a deformation of archetype1 and the other 256 as that of archetype0.

With the four cited chromosomes we get four reclassifications that were pasted into a calc sheet to be compared. We found that the different outputs complement each other. The whole ensemble was able of getting 200 matches without a single mismatch that were distributed in classes according to the BBB distance to the archetypes as follows:

Distance	Archetype1	Archetype0
0	1	1
1	9	9
2	36	36
3	48	48
4	6	6
5	0	0
Total	100	100

In the first column we see the BBB distance to each one of the archetypes. In the second, the number of bitmaps that the new architecture was able of correctly classify without contradictions as deformation of archetype1. The third column shows the corresponding values for archetype0.

The corresponding table produced by the old architecture is, according to program `compareTwoClassifications3.html` of our *Intro*, the following

Distance	Archetype1	Archetype0
0	1	1
1	9	9
2	36	36
3	84	84
4	0	0
5	0	0
Total	130	130

The numbers of bitmaps that are correctly classified coincide for distances 0, 1 and 2. For distance 3, the old architecture is better (it correctly classifies all 84 possible instances, while the new one only 48), for distance 4 the new is better, 6 against 0. Thus, what we established in the previous exercise is false: the old architecture does not always fire when the new one does it. Nevertheless, when the

new architecture fires, the old also does it in a high proportion, more than 90%.

13, page 19. The source of the confusion is that our aim, consigned in the score function of the Evolutionary Process, is incompatible with the alleged property of the addition of inequalities. The task given to ourselves and to Evolution was to design an NN that classifies bitmaps in one of two classes maximizing the number of matches but with no mismatch or contradiction. We got three subtasks represented by three inequalities and recalled that inequalities of real numbers can be added together. In that way, we predicted that the old architecture fires when the new does and therefore the old architecture generalizes the new one. But direct counting evidenced that this false. Why? The problem is that the result of adding inequalities takes care of coincidences but not of contradictions or mismatches, as explicitly stated in the score function. Therefore, we can not claim that when the new architecture fires, the old also does it.

14, page 20. The Scientific Method is a procedure to test beliefs through comparison of their predictions with the observed experimental results and with clarifying discussions. If predictions are fulfilled and resists ruthless discussion, the belief gains credibility and more tests are worth being created. If results contradict the belief, this is abandoned else modified to be tested again.

Our initial belief was the *divide and conquer* strategy of Computer Science to solve problems: tasks can be subdivided to get more tractable, more easily solvable subtasks. In our opinion, this dictum is clearly illustrated by the alphabets that divide the problem of transcribing expressions in transcriptions of phonemes that must be joined together. We aspired to do the same with graphics, by dividing a grid where graphics are drawn into subgrids.

To test our belief, we built two NNs. The old one had two percep-

trons and was targeted to recognize patterns drawn in a 9×9 -grid. The new one was targeted to recognize subpatterns in each row of the grid, 3 subproblems at all.

Based on the idea of the alphabets, we tacitly imagined that the new NN shall be more powerful than the old one, i.e., it correctly fires more times than the old one. This resulted to be false: by direct counting, we found that while the old one was able of recognizing, without mismatches, 260 patterns, the new one was able of recognizing only 128 also without mismatches. This shows that there are many instances when the old NN fires but the new one not. We see that the prediction is false: the new architecture is not better than the old one.

Next, by adding inequalities that characterize sub-RNs, we predicted that the old RN must generalize the new one, i.e., if the new RN fires, the old one must also fire. Direct counting showed that this is also false because the new RN detected 6 bitmaps for distance 4, but the corresponding number was 0 for the old RN. Therefore, our final verdict is that **no NN generalizes the other**. Nevertheless, the old RN statistically generalizes the new one and is cheaper.

We also got a prediction that was verified by observation: if the total additional bias is 2, we can divide it in three parts as $0+1+1$, $1+0+1$, or $1+1+0$, we say that the division is degenerate. This implies that there must be various chromosomes for the new architecture that render the maximal observed value, 128. We found 4 of them, which together were able of recognizing 200 bitmaps, an improvement of nearly 50%. However, it is not enough to be interesting because we remain very far from the maximal possible value, to recognize 512 bitmaps out of 512.

8 THE BIBLIOGRAPHY

References

[1] Divide and Conquer Algorithm (2024)
https://www.geeksforgeeks.org/divide-and-conquer/ 3

[2] José Rodríguez (2024)
Evolutionary Neural Networks 1. Introduction.
https://www.ejristos.com/Evol%20Neural%20Networks/EvolNNintro.pdf
https://www.ejristos.com/Evol%20Neural%20Networks/EvolNNintroSpanish.pdf 3

[3] José Rodríguez (2024)
Evolutionary Neural Networks 2. Simplification.
https://ejristos.com/Evol%20Neural%20Networks/EvolNN2.pdf
https://ejristos.com/Evol%20Neural%20Networks/EvolNN2Spanish.pdf 3

These links were active by 15/Nov/2024