Faculty of Technology and Bionics

Applied Research Project

Report

Introduction to Artificial Intelligence via Neuroevolution

Author: Marcello Tania
E-Mail: marcello.tania@hsrw.org
Supervisor: Prof. Achim Kehrein

Submission Day: 12th January 2023

Contents

1 Introduction

I have always been curious about how a machine learns by itself, and I have been lucky to have taken an Evolutionary Algorithm class by Prof. Achim Kehrein. His class gave me a basic knowledge of how to do optimization problems. During the winter holiday in 2020, I got a chance to start a project in Python just for fun. I applied my knowledge from that class, which was insufficient to solve the problem. So, I needed to learn more about neural networks. I found out that the combination of an Artificial Neural Network and Evolutionary Algorithm is called Neuroevolution. Finally, combining Neuroevolution with my digital fabrication expertise to create a small clicker robot. The project is becoming like a machine-to-machine interaction which the small robot clicks the space key on an ordinary keyboard. The robot worked well at the very end, and I was impressed with the result. I had never thought the robot could learn and improve due to the training progress. The robot ends up being a better player than me.

This report will describe how to implement Evolutionary Algorithms and Artificial Neural Networks through designing, constructing, and programming machines that learn how to play Flappy Bird. When we understand how to code from scratch and experience how Artificial Intelligence is implemented on our own, we will be able to solve more real-life problems and eventually use a robot. Overall, this report explains my step-by-step learning curve on how to implement a simple game to use AI to play on its own, and lastly, build a robot to think and learn to play the game physically by controlling a keyboard. The experience is incredible to experience how the machine is learning by itself, looking on how the robot click and even double click with the correct timing.

Many problems have been successfully solved using Evolutionary algorithms, but the full potential has not been reached yet. I would like to see more robots learn independently and do some practical tasks.

My general research topic is: How can we implement a robot and Artificial Intelligence via Evolutionary algorithms to real-life applications?

2 Problem Statement: Flappy Bird

Flappy Bird is a mobile video game app developed by Dong Nguyen in 2013. The goal of this game is that the player needs to guide the bird agent flying with a constant horizontal velocity component through gaps between vertical pipes at various heights. The player controls the height of the bird by doing nothing to let the bird gradually fall due to gravity or by tapping the screen to make the bird flap its wings, and thereby the bird will move up. The game ends when the bird hits a pipe or the ground. [1, 2]

3 Theoretical Foundations

3.1 Evolutionary Algorithm

3.1.1 What is Evolution?

To explain the evolutionary algorithms, perhaps we will begin by introducing Darwinian Evolution in terms of biology. Looking at evolution in terms of biology is straightforward because of the evidence that animals evolve, and the story is easy to imagine.

The theory of evolution has become essential to explain many things, and Darwin was awarded the Nobel Prize. Darwinian Evolution is the red string in biology. Evolution equips a robust framework for understanding life. The story begins with Charles Darwin, who wrote the book "On the origin of species" in 1859. When Darwin was observing nature, he found the four fundamental concepts:

1. "one or more populations of individuals competing for limited resources,
2. the notion of dynamically changing populations due to the birth and death of individuals,
3. a concept of fitness that reflects the ability of an individual to survive and reproduce, and
4. a concept of variational inheritance: offspring closely resemble their parents, but are not identical."[3]

Limited resources mentioned in the first point may refer to a lack of food, lack of habitat, and lack of mates. As a result, not all individuals will survive or reproduce, which leads to the second point. The second point is when individuals are replaced by younger offspring and reproduce to form the next generation. The third observation is where there are competitions. Some traits are more advantageous than others. "The survival of the fittest," the fitter individuals are more likely to survive and reproduce. Last but not least, parents pass their traits to their offspring, noting that offspring have different traits from parents, such as hair color, eye color, height, etc. The trait is defined as 'phenotype.' The offspring and the parents may look similar but never identical.

All in all, the percentage of advantageous traits in a population increases from generation to generation. "natural selection" explains how populations could evolve in such a way that they become better suited to the environment over time.

The theory of evolution has been proven with many examples of discoveries. First is what happened to Galapagos finches. You can find the same bird that has different finches in different places. You might ask a question: How can the same bird has different finches depending on the place? Of course, the Darwinian evolution and the bird evolve over billions of years. One has a long and sharp beak to eat cactus flowers and pulp, the other that has a narrow and pointed beak to grasp insects,s and the last one that has a large ground finch to crack seeds.[4] Second, a story about a stick insect that evolves to camouflage from the bird predator. Not only the insect evolves to look like a stem, but the bird's eye also improves. [5] Last but not least, HIV has the ability to reproduce rapidly into a new different virus. When the 3TC drug is given to stop the reproduction of HIV virus in a patient, the next generation of HIV virus leads to a drug-resistant

variant virus which became a big problem in the 90s. [6] Similar reason why antibiotics should always be taken until the end, even if patients feel better after two or three days. The antibiotics need to kill all the bacteria and not only reduce the growth and reproduction of a bacteria. One alive bacterium is possible to become "stronger" and immune to the antibiotic.

Figure 1: Cactus-eater

Figure 2: Insect-eater

Figure 3: Seed-Eater

Figure 4: Specification of Galapagos Finches [4]

3.1.2 Problem Set: Absolute Maximum

After getting the basic ideas in biology, we will start translating them into code. We will apply the Darwinian Evolutionary observations to solve a simple problem to understand Darwinian Evolution.

We begin with a simple problem, find the absolute maximum of the following function:

$$\begin{aligned} f : [-2, 3] &\to \mathbb{R}, \\ x &\mapsto 2x^3 - 3x^2 - 12x + 1. \end{aligned} \tag{1}$$

To better understand the function, a plot of the function might be helpful. To graph the function, we need two libraries: Numpy to work with math and Matplotlib to plot the function. The following are the Python codes used to plot function (1).

```
import numpy as np # import Numpy library and simplified it as np
import matplotlib.pyplot as plt # import Matplotlib.pyplot and
   simplified it as plt


def f(x):
  return ((2.0 * x - 3.0) * x - 12.0) * x + 1.0;


x = np.linspace(start=-2, stop=3, num=200) # return 200 samples
   discretly from range of -2 to 3
plt.plot(x, f(x)) # plot x and f(x)
```

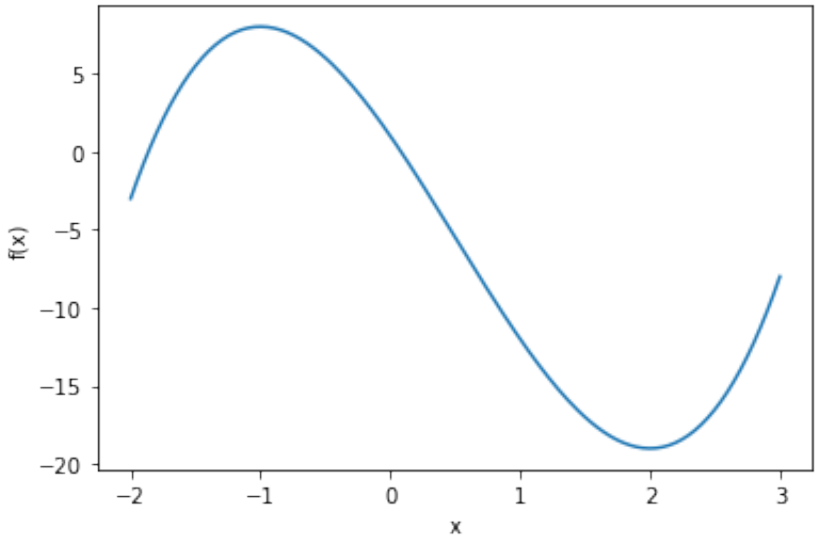


Figure 5: Plot of function (1).

From Figure 5, we can see that the absolute maximum of f is reached when x lies around -1, but this guess is not accurate enough. We will follow the strategy of evolution to improve our guess of the maximum value. The following is a pseudo-code of the so-called "evolutionary algorithm," an algorithm that is inspired by the Darwinian evolution theory.

```
EA:
    0.  Represent the individuals.
    1.  Generate randomly initial population.
    Do Forever:
        2.  Choose Parents.
        3.  Select parents to reproduce a single offspring
        4.  Compute the fitness of the offspring
        5.  Select a member of the individuals to die, "Struggle for
   Survival"
```

Listing 1: Pseudo Code of Evolutionary Algorithm (EA).

First, we need to take a uniform random sample of our domain (denoted by x_i), which is the interval $[-2, 3]$. We will define these numbers x_i as individuals. These individuals are what we call the initial population. The strategy is to evaluate the "fitness" of each individual x_i, which is nothing else but $y_i = f(x_i)$. The higher the fitness is, the better the individual is because we want to find the global maximum. According to Rule 3, individuals with low fitness might not survive and reproduce (less probability). They might then get replaced by the "offsprings," which, in this case, is a different sample in the interval $[-2, 3]$. All the individuals that have a low value of fitness are replaced, and the ones that have a high value of fitness "survive." This process is repeated until we stop the program, and the individual with the highest fitness is kept. This individual is the one that we consider as the optimal value of x.

3.1.3 0. Representation of Individuals

The first question we have to answer is, "How can we represent an individual?". Recall that the problem is to find the global maximum where the x value is given to is the maximum value of y. Here we would have to choose the representation of the solution candidates. The solution can be found by simply choosing the highest function value f(x) for the best individual x. The search space is in between -2 and 3. However, computers cannot handle infinitely many numbers, so we would have to discretize them. eg -2,00, -1.99, .., 3.00. All individuals must have what we call "phenotype" and "fitness." The phenotype represents a particular trait of an organism. Characters represent the individual. Think about traits that are easily observed, for instance, hair color, eye color, skin color, height, and weight. The traits are what we call "phenotypes," and the number of phenotypes in this example is only one. Another important term is called "fitness." Fitness is defined as the function f(x) in this problem. Fitness depicts how good the individual is.

```
import numpy as np
import matplotlib.pyplot as plt


def pheno2fitness(ph):
  return ((2.0 * ph - 3.0) * ph - 12.0) * ph + 1.0;

x = np.linspace(start=-2, stop=3, num=200) # population range
plt.plot(x, pheno2fitness(x))
```

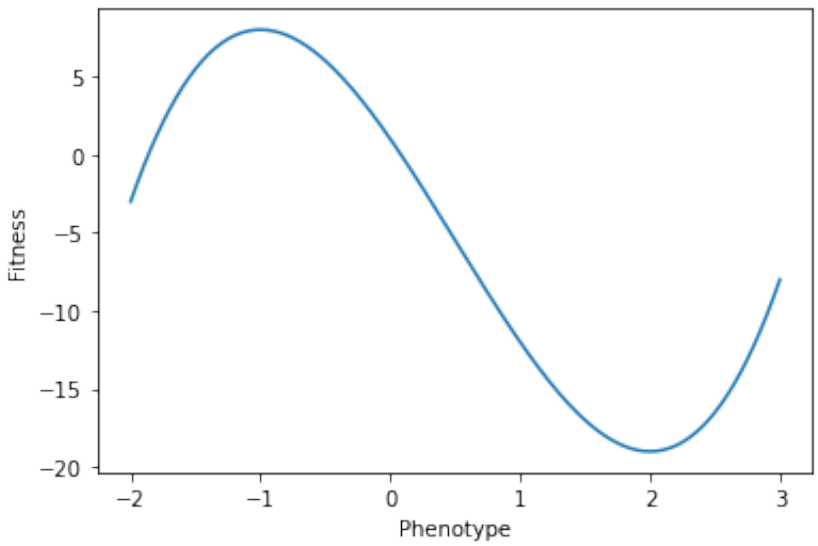


Figure 6: Phenotype vs fitness

3.1.4 1. Create an Initial Population

Depending on the interest, most problems could be solved with simple evolutionary models that focus on the evolution over time with fixed population size in a fixed environment and mechanism for reproduction and inheritance. [3]

The first population is often generated randomly over the search space. In this example, ten individuals are uniformly created between the number of [-2,3]. The total number of individual is ten.

Table 1: First Population

Index	Phenotype	Fitness
0	PH[0]	f(PH[0])
1	PH[1]	f(PH[1])
2	PH[2]	f(PH[2])
.	.	.
.	.	.
.	.	.
9	PH[9]	f(PH[9])

Ten individuals are created using randomness from -2 to 3. Phenotype is the x, and fitness is the y. We are defining that there will be a fixed number of 10 individuals in every generation.

```
POP_SIZE = 10 # defining population size
pheno = [None] * POP_SIZE
fitness = [None] * POP_SIZE
```

Let us try to create the first individual using randomness and then print the number.

```
pheno[0] = random.uniform(-2, 3) # return random float point number
print(pheno[0])
```

```
Output :
2.6827079798837454
```

Now, let us also translate the phenotype into the fitness of the first individual using the function we had created before.

```
fitness[0] = pheno2fitness(pheno[0])
print(fitness[0])
```

```
Output :
-18.963846797652124
```

If everything is fine, repeat it ten times, so we will create ten individuals.

```
print('Generation 0')
print('{}\t| {}\t\t| {}'.format('Index', 'Phenotype', 'Fitness'))
print('-------------------------------------------------------')
for indiv in range(POP_SIZE):
```

```
    pheno[indiv] = random.uniform(-2, 3)  # return random float point
    number
    fitness[indiv] = pheno2fitness(pheno[indiv])
    print('{}\t| {}\t| {}'.format(indiv, pheno[indiv], fitness[indiv]))
```

```
Output :
Generation 0
Index | Phenotype    | Fitness
-----------------------------------------------------------
0 | -0.7641674171551771 | 7.525679540702386
1 | 1.0126653783118797  | -12.151499240966746
2 | 0.9496898463608883  | -11.388939502875928
3 | 0.8498101931950743  | -10.136827040492891
4 | 0.9121184058467935  | -10.92360879617874
5 | 0.7967986566050973  | -9.454492207741476
6 | 1.690114031485857 | -18.195252452290667
7 | -1.9410882278453676 | -1.637767503737373
8 | 0.20075837405615893 | -1.5138295621316669
9 | -1.608365804543757  | 4.218696169883673
```

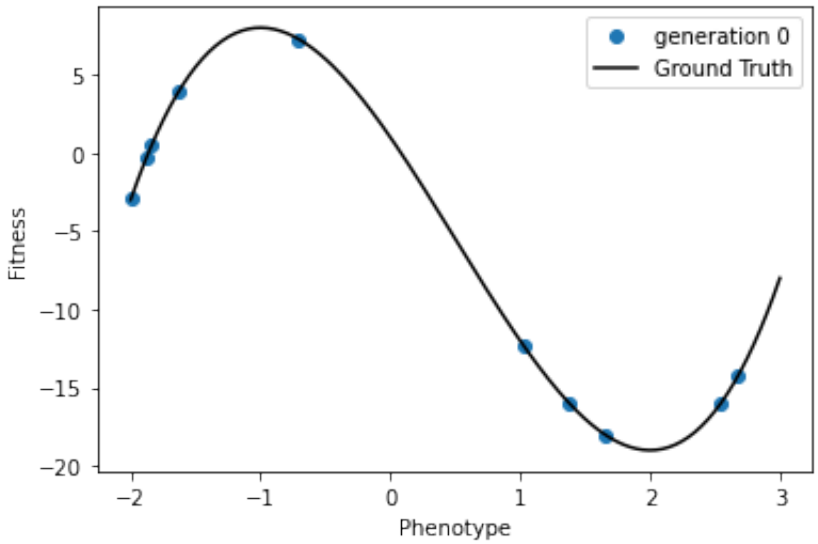


Figure 7: Generation 0.

Congratulations! You have created the first generation of 10 individuals. Next, we also have to do a few statistics. Keeping the best individual and average fitness for every generation. Keep in mind that five is for future reference that we are going to simulate till the 5 generations. Now is just the first generation.

```
MAX_GEN = 5 # define the maximum generations

best_fitness = [None] * MAX_GEN
average_fitness = [None] * MAX_GEN

generation = 0
```

```
best_indiv = 0
total_fitness = 0
best_fitness[generation] = fitness[0]
```

Before we just created ten individuals, now, we would need to update the statistical analysis. Therefore, we added some codes to have the average fitness, best fitness, and best individual. Update statistical analysis

```
print('Generation 0')
print('{}\t| {}\t\t| {}'.format('Index', 'Phenotype', 'Fitness'))
print('-------------------------------------------------------')
for indiv in range(POP_SIZE): # Iterate for every individual
    pheno[indiv] = random.uniform(-2, 3) # return random float point
    number
    fitness[indiv] = pheno2fitness(pheno[indiv])
    print('{}\t| {}\t| {}'.format(indiv, pheno[indiv], fitness[indiv]))

    # update statistical analysis
    total_fitness += fitness[indiv];
    if fitness[indiv] > best_fitness[generation]:
        best_fitness[generation] = fitness[indiv];
        best_indiv = indiv

average_fitness[generation] = average_fitness[generation] / POP_SIZE;
```

```
Output :

Generation 0
Index | Phenotype    | Fitness
-------------------------------------------------------
0 | 1.0845787880645643  | -12.992274661806283
1 | -1.7218151438788802 | 2.558690013813912
2 | 0.7183167787332914  | -8.426465590024545
3 | 0.35281459643526336 | -3.519374166520877
4 | 2.627364070270314 | -14.963885885778055
5 | 1.5782533926206446  | -17.549200499988967
6 | 1.9110663673314066  | -18.930224067001
7 | -1.875916207006643  | -0.2491197997823127
8 | 0.44130965305977066 | -4.708084640777012
9 | -1.5013083121648212 | 5.486242177621906
```

Print statistical analysis.

```
print("The average fitness is {}".format(average_fitness[generation]))
print("The best fitness is {}".format(best_fitness[generation]))
print("The best individual is {}".format(best_indiv))
```

```
Output :
The average fitness is -7.329369712
The best fitness is 5.486242177621906
The best individual is 9
```

3.1.5 2. Select Parents and create offspring

The first generation is done. Next is to create the next generation. How to do that? Parents need to create kids. The kids are generated by combining parental information and then a bit of mutation. The parent is randomly chosen.

```
parent = random.randint(0, 9) # return any random integer from 0 to 9

print("The chosen parent is {}".format(parent))
```

```
Output :
The chosen parent is 0
```

3.1.6 3. Create offspring mutation of traits

To create offspring is by choosing one parent. This process is not biologically standard, but we want to keep it simple at the moment. Therefore, we choose one parent and create a slightly different offspring. In this case, the offspring can be -0.1 or +0.1 different.

```
STEP_SIZE = 0.1

rival = random.choice([-1, 1]) # choose single number from the list
    numbers -1,1
offspring_phenotype = pheno[parent] + (rival * STEP_SIZE)  #
    calculating offspring phenotype
```

A problem with the mutation is that we must ensure it is within the search space. So we created a strategy called bounce back.

3.1.7 Bounce Back

If the offspring phenotype is more than three and less than -2, the offspring phenotype will bounce back to the search space.

```
if offspring_phenotype > 3:
    offspring_phenotype = 3 - (offspring_phenotype - 3)
if offspring_phenotype < -2:
    offspring_phenotype = -2 - (offspring_phenotype + 2)
```

Okay, you will have to be able to get the offspring right now.

```
print("The phenotype of the offspring is {}".format(offspring_phenotype
    ))
```

```
Output :
The phenotype of the offspring is 2.4122288402456387
```

3.1.8 Compute the fitness of the offspring

After producing the offspring, you also need to get fitness.

```
offspring_fitness = pheno2fitness(offspring_phenotype); # calculating
    offspring fitness

print("The fitness of the offspring is {}".format(offspring_fitness))
```

```
Output :
The fitness of the offspring is -17.330504198398376
```

3.1.9 4. Struggle of Survival

Selection to form the next generation, which includes parent population and offspring. In a natural evolution, there is an age where old individuals might die, but we would still keep the best and strongest individual to the next generation. This will give us a better chance to get the best solution.

3.1.10 5. Select member of the population to die

Rival vs Offspring

```
rival = random.randint(0, 9) # return any random integer from 0 to 9

print("The chosen rival is {}".format(rival))
print("The fitness of the rival is {}".format(fitness[rival]))
```

```
Output :
The chosen rival is 8
The fitness of the rival is -18.760975010104296
```

```
if offspring_fitness > fitness[rival]:
    pheno[rival] = offspring_phenotype;
    fitness[rival] = offspring_fitness;
    print("Rival is being replaced with the offspring!")
```

```
Output :
Rival is being replaced with the offspring!
```

Replaced to the new generation

```
generation = 1
total_fitness = 0
best_fitness[1] = best_fitness[0]

print('Generation 1')
print('{}\t| {}\t\t| {}'.format('Index', 'Phenotype', 'Fitness'))
print('-------------------------------------------------------')
for indiv in range(POP_SIZE):
    fitness[indiv] = pheno2fitness(pheno[indiv])
    print('{}\t| {}\t| {}'.format(indiv, pheno[indiv], fitness[indiv]))

    # update statistical analysis
    total_fitness += fitness[indiv]
    if fitness[indiv] > best_fitness[generation-1]:
```

```
        best_fitness[generation] = fitness[indiv]
        best_indiv = indiv
```

```
Output :
Generation 1
Index | Phenotype    | Fitness
-------------------------------------------------------
0 | 2.3122288402456386  | -18.061741950463944
1 | 2.23635968399351  | -18.470798004999548
2 | 2.7191878069050555  | -13.60094748234499
3 | 2.0180034517840513  | -18.997071210803178
4 | 0.1725640148979446  | -1.149825856936729
5 | -1.0748728375078818 | 7.948707058293252
6 | 1.2367891578345227  | -14.646709465776443
7 | 1.2540709229121396  | -14.822393383695275
8 | 2.4122288402456387  | -17.330504198398376
9 | -0.6195250891599291 | 6.807305557189119
```

Statistics

```
average_fitness[generation] = total_fitness / POP_SIZE;

print("The average fitness is {}".format(average_fitness[generation]))
print("The best fitness is {}".format(best_fitness[generation]))
print("The best indiv is {}".format(best_indiv))
```

```
The average fitness is -10.23239789379361
The best fitness is 7.948707058293252
The best indiv is 5
```

3.1.11 Codes

Full codes

3.1.12 Result

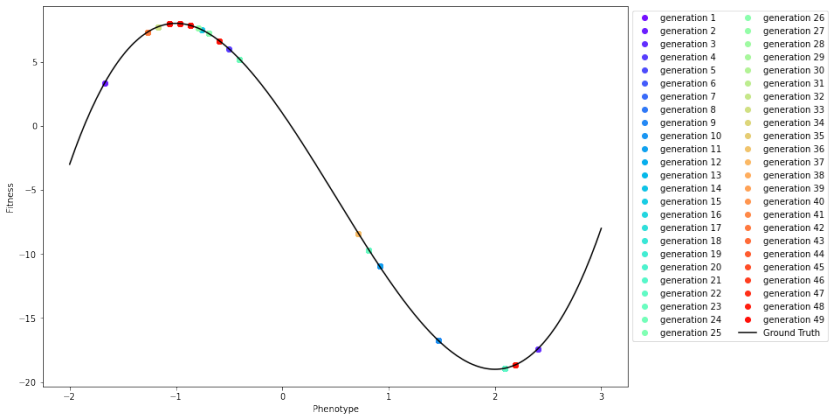


Figure 8: Fitness over a generation.

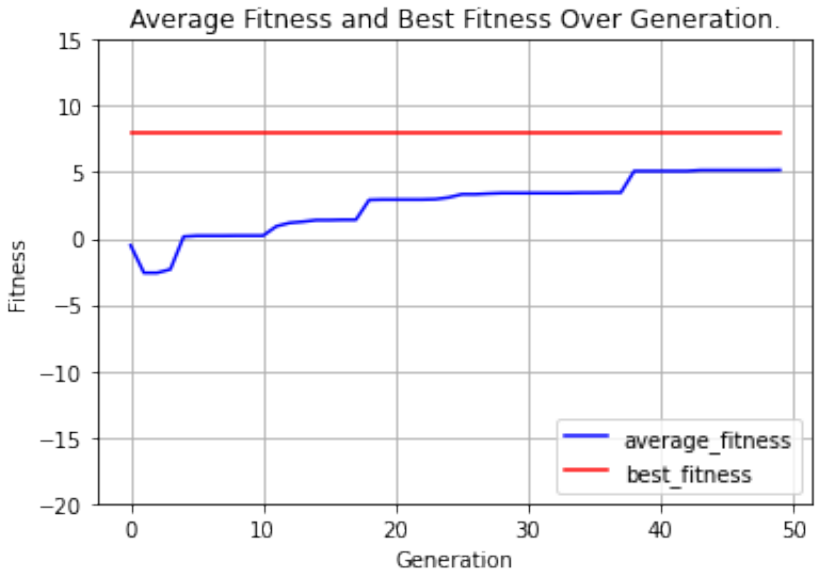


Figure 9: Summary of evolutionary algorithm.

3.2 Genetic Algorithm

3.2.1 What is Genotype?

DNA is what stores the blueprint of our traits (the phenotype). Most of our traits are inherited from our parents, and the collection of all traits is called the phenotype. Genotype is a collection of all genes of an individual. Therefore, the genotype determines the phenotypes. Those genes are the information that programs all of our cell activity. It is a 6-billion letter code that provides the assembly instructions for everything you are.

In sexual reproduction, both genotypes of two parents are combined for reproduction.

All in all, imitating these ideas leads to a class of evolutionary algorithms called genetic algorithms.

The DNA has a sequence of four nucleotides: Adenine, Thymine, Guanine, and Cytosine.

3.2.2 Representation of Genotype

This uses some code, and we must translate the process to the phenotype. The researcher John Holland introduced a genetic algorithm using the binary alphabet 0 and 1 to describe the genotype.

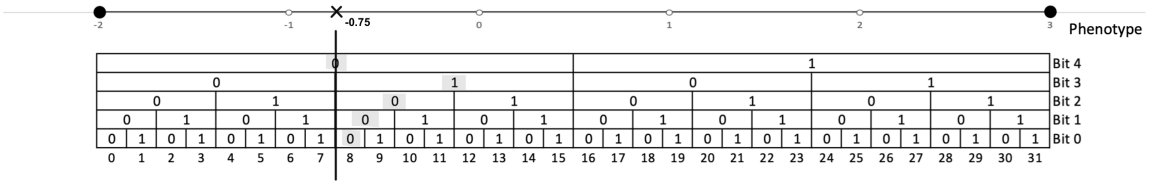


Figure 10: Genotype to Phenotype

For instance, we have a genotype of 01000 with an interval between -2 to 3 of 4 bits. To translate the genotype, which is binary, into decimal these are the equation:

$$\frac{1}{\text{total number of intervals}} \cdot (\text{number of interval}) \cdot \text{width of interval} + \text{left end point of interval} \tag{2}$$

$$\frac{1}{2^5} * (Bit_4 \cdot 2^4 + ... + Bit_0 \cdot 2^0) \cdot 5 - 2 \tag{3}$$

Recall how to translate binary to decimal of 4 bits: [1]

$$Bit_4 \cdot 2^4 + Bit_3 \cdot 2^3 + Bit_2 \cdot 2^2 + Bit_1 \cdot 2^1 + Bit_0 \cdot 2^0 = decimal \tag{4}$$

$$\frac{1}{32} * (0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0) \cdot 5 - 2 = -0.75 \tag{5}$$

3.2.3 Mutation

In evolution, they have what we call a mutation. A mutation is a slight change in the genome. One of the ideas is that flip a bit. De Jong suggested using a small probability to decide whether each bit is going to be flipped or not. This mutation will affect the search space, significant change will jump into a different search, and a small change will give a more precise search into most of the population.

[1] `https://cllom.gitlab.io/mynotes/ComputionalThinking`

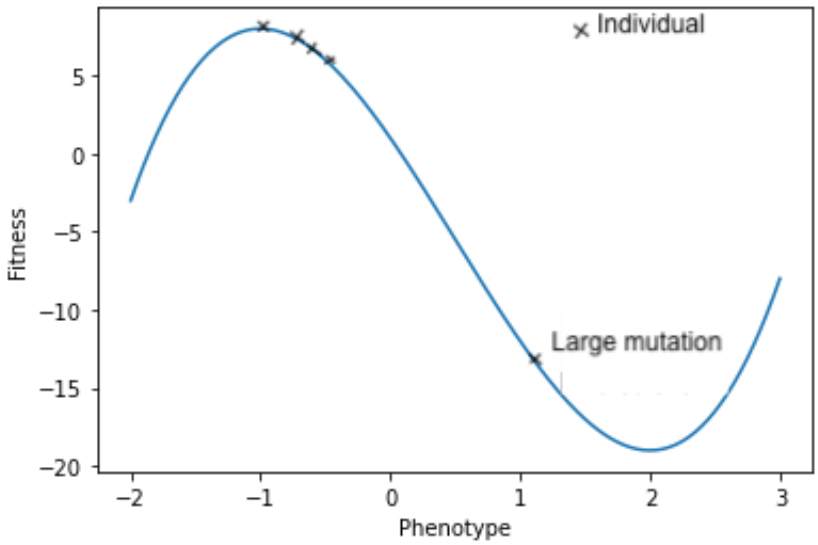


Figure 11: How does mutation affect the search space?

3.2.4 Recombination(Sexual reproduction)

In reproduction, we would have mom and dad. One of the methods is called a single cross-over. A single cut from mom and also dad, then combine one part from one parent with the other part from the other parent. The location of the cut is generally chosen randomly.

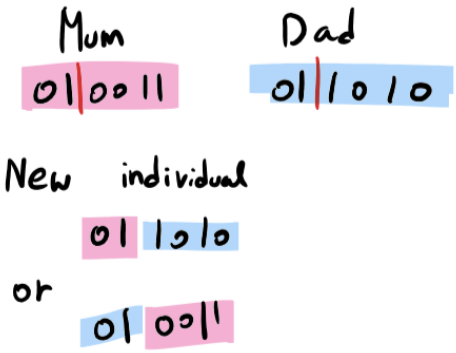


Figure 12: Recombination.

Single crossover is not the only option; two-point cross-over or three-point cross-over can be an alternative.

There is also a uniform crossover. For each bit(gene), we decide randomly from which parent it will be inherited.

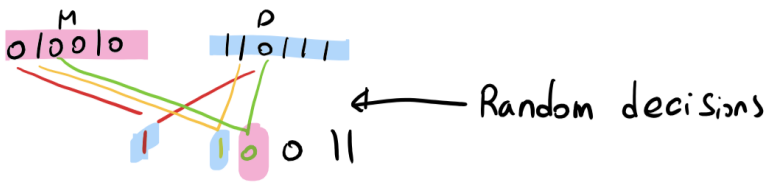


Figure 13: Uniform cross over

3.2.5 Problem Set

The problem set is finding a global maximum, the same problem as in Figure 5.

To code a genetic algorithm, we would need a genotype by representing in terms of bits.

Representation of Individual

```
class Individual():
    def __init__(self, gene, pheno, fitness):
        self.gene = gene
        self.pheno = pheno
        self.fitness = fitness
```

In a genetic algorithm, each individual has a gene that can be represented in binary, and phenotype, in this case, is the decimal value. Fitness is the y-axis.

From Figure 10, converting the genotype 01001 into a phenotype can easily inputting the genes into a function of geno2pheno.

```
def geno2pheno(gene):

  # convert bits to integer
  number = gene[0]
  for i in range(5-1):
    number += gene[i]
    number *= 2.0
    print('i: {}| gene: {} |number: {}'.format(i, number,gene[i]))
  return (number/pow(2,5))*5.0 -2.0 # convert interval number into
   phenotype

gene = np.array([0,1,0,0,1])

print(geno2pheno(gene))
```

```
Output:

i: 0| gene: 0.0 |number: 0
i: 1| gene: 2.0 |number: 1
i: 2| gene: 4.0 |number: 0
i: 3| gene: 8.0 |number: 0
-0.75
```

3.2.6 Codes

Full codes

3.2.7 Result

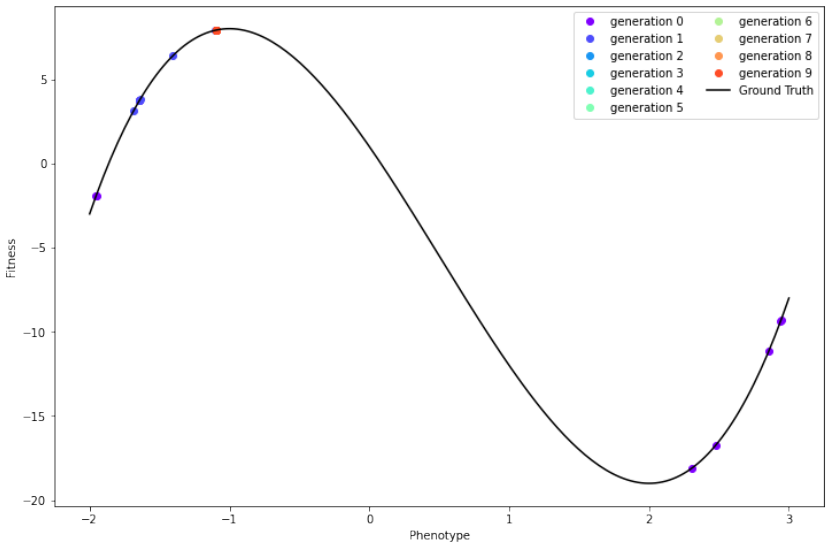


Figure 14: Fitness over generation.

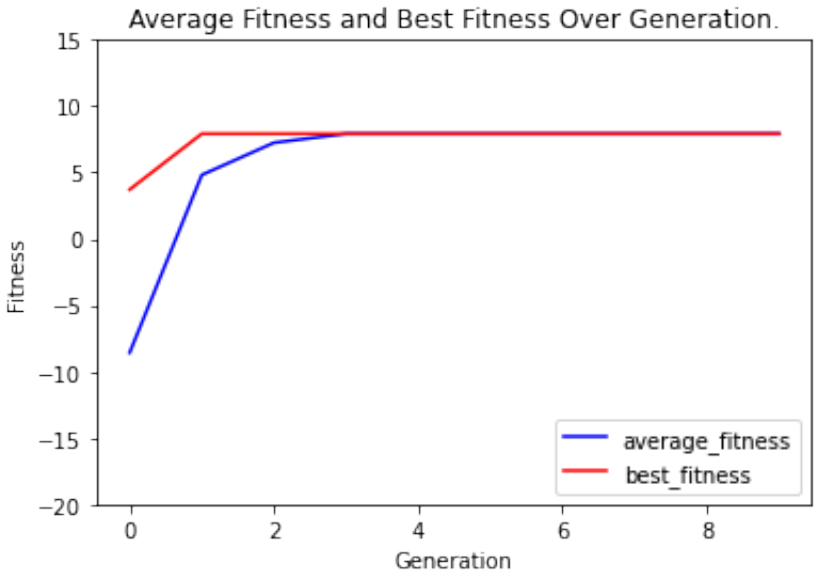


Figure 15: Summary of genetic algorithm.

3.3 Artificial Neural Network

Our brain inspires the basics of Artificial Neural Networks (ANN). The brain consists of billions of cells called neurons that are connected. The interconnected neurons are where the superpower comes from. [7] The artificial neuron that is created in the program is called a node, and to represent how strong is a connection between nodes is called weight.

3.3.1 Neuron

How does a neuron work? Neuron takes an electric input and pops out as another electrical signal. Basically, from the left to right of Figure 16, you can take a look at a dendrite as the input of the electrical signal. Second, going to the axon, and lastly, giving the output signal to the terminal to another neuron.

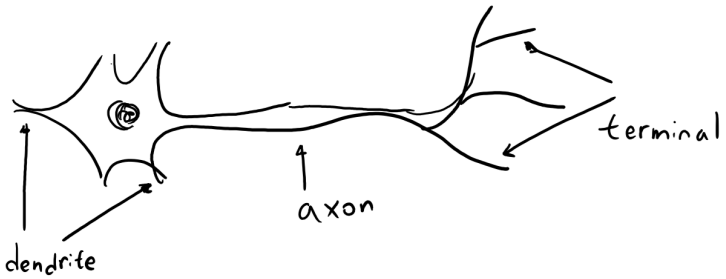
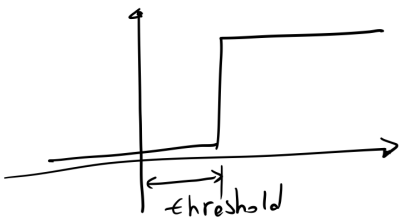
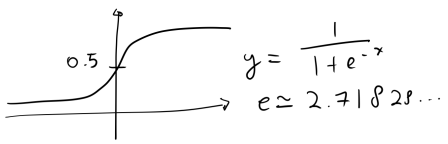


Figure 16: Neuron

However, in the observation, the neuron is only activated with a threshold. In other words, neurons do not react immediately but instead hold in that input until the electric current has reached enough to trigger the output—the neuron fire when the input reaches the threshold, just like a step function. Indeed, in nature is very hard to find a cold hard edge. Hence a sigmoid function is a closer representation. The sigmoid function is one example of what we call **activation function**.



(a) Step function



(b) Sigmoid Function

Figure 17: Activation Functions

In real biological neurons, a single neuron takes many inputs. The sum of the input

x is summed, and if the sum of x is large enough, an electrical signal from the dendrites are combined and fired to the axon.

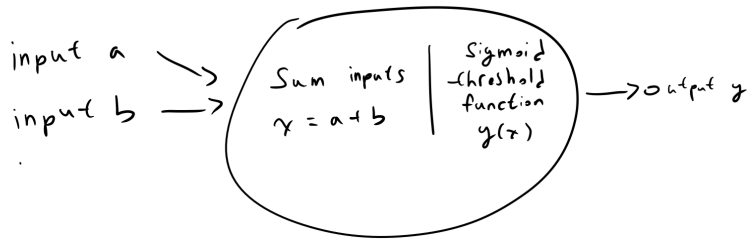


Figure 18: Two inputs for a boolean logic machine.

Although an individual neuron processes simple information, the power comes from 100 billion neurons interconnected inside our brain. Each neuron has roughly 1000 to 10000 connections with another neuron. As a result of this fully interconnected neuron is an intelligent brain that can learn. [8]

Figure 19: Neural Network .

The architecture of ANN has three layers: the input layer, the hidden layer, and the output layer. Typically, the number of the hidden layer can be more than one layer or none.

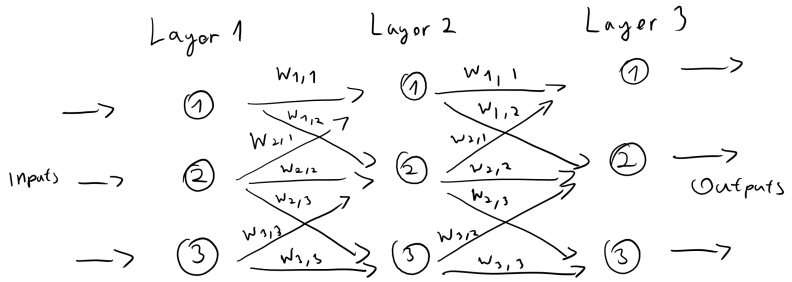


Figure 20: Representation of neural network.

3.3.2 Make your own ANN

To make things clear, let us start with a straightforward ANN, with two layers and four neurons. Each neuron has **weighted input**, activation function(some papers called it also transfer function) and one output'. The weight is an adjustable parameter to getting to know which inputs are 'important'.[8] The weight of the inputs is where to get to know how important the input is going through the neuron.

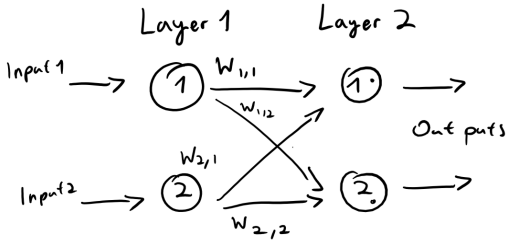


Figure 21: Example ANN problem.

$$
\begin{aligned}
w_{1,1} &= 0.9 \\
w_{1,2} &= 0.2 \\
w_{2,1} &= 0.3 \\
w_{2,2} &= 0.8 \\
x_1 &= (1 \cdot 0.9) + (0.5 \cdot 0.3) \\
&= 0.9 + 0.15 = 1.05 \\
y_1(1.05) &= 0.7408 \\
x_2 &= (1 \cdot 0.2) + (0.5 \cdot 0.8) \\
&= 0.2 + 0.4 = 0.6 \\
y_2(0.6) &= 0.6457
\end{aligned}
\tag{6}
$$

The previous example only shows us four neurons, yet the brain consists of hundred

of millions of neurons. To express that, matrix multiplication comes in handy. We could express it into numbers where the size of the neural network does not matter.

$$\begin{aligned} x_1 &= (input1 \cdot w_{1,1}) + (input2 \cdot w_{2,1}) \\ x_2 &= (input1 \cdot w_{1,2}) + (input2 \cdot w_{2,2}) \\ x &= wi \\ &= \begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} input_1 \\ input_2 \end{pmatrix} \\ &= \begin{pmatrix} (input1 \cdot w_{1,1}) + (input2 \cdot w_{2,1}) \\ (input1 \cdot w_{1,2}) + (input2 \cdot w_{2,2}) \end{pmatrix} \\ O &= sigmoid(x) \end{aligned} \tag{7}$$

Now, we add one hidden layer with three neurons. Again each neuron has a weighted input, activation function, and one output. This network proceed input signals from the dendrites, multiplied by the adjustable weight and then passes through the transfer function to produce the output. The network is called **feedforward network**, the network's output does not have a relation to the input.

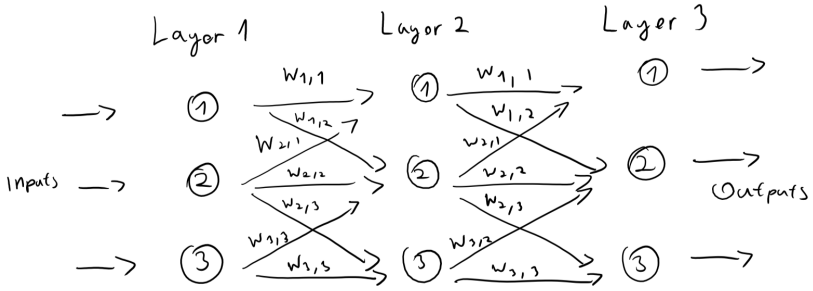


Figure 22: Three layers of matrix multiplication.

$$\begin{aligned} x_{hidden} &= w_{input_hidden} \cdot i \\ O_{hidden} &= sigmoid(x_{hidden}) \\ x_{output} &= w_{hidden_output} \cdot O_{hidden} \\ O_{output} &= sigmoid(x_{output}) \end{aligned} \tag{8}$$

3.4 Neuroevolution

The feedback network is where the output signals are kept and used to minimize the training error. Instead of backpropagation, we would use a genetic algorithm to search for the appropriate weights connecting all the neurons. Each cycle of feedforward and feedback tries to change the value of the weight until the networks reach a certain amount of accuracy.

3.4.1 Problem Set: XOR

The concept seems too abstract, so practically is very good again to have a simple problem and apply the knowledge. Generally, many papers and books introduce back-propagation. Rumelhart, Hinton, Williams, and Parker separately methodized back-propagation networks. To learn more about backpropagation, look at Henseler's paper [9]. Since, we are learning neuroevolution, We adjust the weights by using an evolutionary algorithm.

```
# XOR problem
import numpy

data_inputs = numpy.array([[1, 1],
                           [1, 0],
                           [0, 1],
                           [0, 0]])

data_outputs = numpy.array([0,
                            1,
                            1,
                            0])
```

The artificial intelligence we are about to create to solve the XOR problem is based on a knowledge-based system. The knowledge is based on the defining rules. In this case, we created all the possible inputs and outputs according to the XOR rule. This data set will guide the decision of our ANN algorithms.

3.4.2 Codes

Full codes

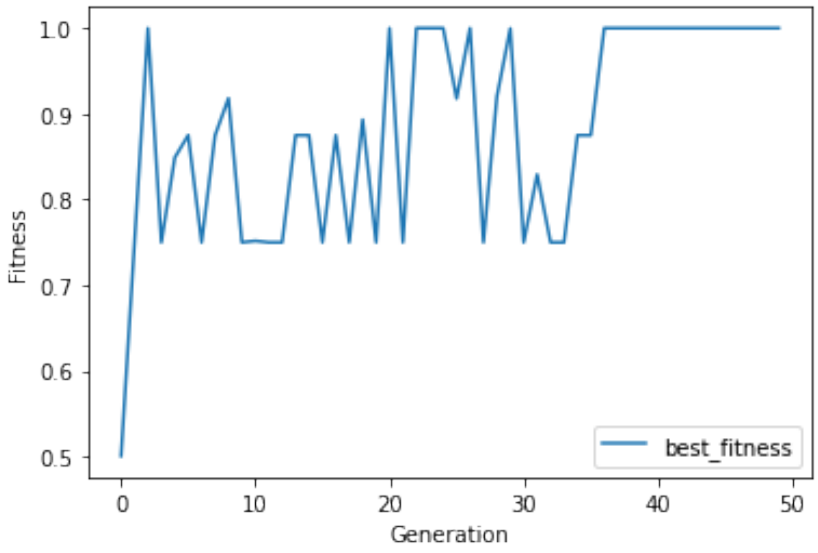


Figure 23: Best Fitness Over Generation.

We are translating the final output using all the weights and inputs.

```
def sigmoid(x):
    s=1/(1+numpy.exp(-x))
    ds=s*(1-s)
    return s
# query the neural network
def query(wih1, wih2, who, inputs_list):
  # convert inputs list to 2d array
  inputs = numpy.array(inputs_list, ndmin=2).T
  # calculate signals into hidden layer
  hidden_inputs1 = numpy.dot(wih1, inputs)
  # calculate the signals emerging from hidden layer
  hidden_outputs1 = sigmoid(hidden_inputs1)
  # calculate signals into hidden layer
  hidden_inputs2 = numpy.dot(wih2, hidden_outputs1)
  # calculate the signals emerging from hidden layer
  hidden_outputs2 = sigmoid(hidden_inputs2)
  # calculate signals into final output layer
  final_inputs = numpy.dot(who, hidden_outputs2)
  # calculate the signals emerging from final output layer
  final_outputs = sigmoid(final_inputs)

  return final_outputs

for x in range(4):
  print(query(best_so_far.wih1,best_so_far.wih2,best_so_far.who,
    data_inputs[x]))
```

```
Output:

[[1.22125201e-44]]
[[1.]]
[[1.]]
[[5.47922475e-47]]
```

4 Methods

4.1 Software

https://github.com/cllom/Machine-Playing-Flappy-Bird

4.1.1 Game Parameters

- Bird ID is the identification of an agent. Score is the score of the current agent playing.
- High score is the highest score for all the agents whom who has played the game.
- Generation is the current generation of all the agents being played.

4.1.2 Artificial Neural Network

Let us call the robot an agent, and each agent has its brain of an artificial network of 4 inputs, seven hidden nodes, and one output.

```
self.wih = numpy.random.normal(0.0, pow(self.hnodes,-0.5), (self.hnodes
    , self.inodes))
self.who = numpy.random.normal(0.0, pow(self.onodes,-0.5), (self.onodes
    , self.hnodes))
```

Then, the forward propagation calculates the output result by using the weights of each node of the Artificial Neural Network. We could just simply calculate this using matrix.

```
def query(self, inputs_list):
    # convert inputs list to 2d array
    inputs = numpy.array(inputs_list, ndmin=2).T
    # calculate signals into hidden layer
    hidden_inputs = numpy.dot(self.wih, inputs)
    # calculate the signals emerging from hidden layer
    hidden_outputs = self.activation_function(hidden_inputs)
    # calculate signals into final output layer
    final_inputs = numpy.dot(self.who, hidden_outputs)
    # calculate the signals emerging from final output layer
    final_outputs = self.activation_function(final_inputs)

    return final_outputs
```

4.1.3 Genetic Algorithm

The evolutionary algorithm or genetic algorithm helps to select the better agents who are more likely survive by using the techniques as follows: survival of the fittest, reproduction, and mutation.

1. First Population

```
# Create first population
print('Geration : 0, HELLO WORLD!')
print('Indiv\twho\tFitness')
print('-------------------------------------------------')

for i in range(POP_SIZE):
    person[i] = Individual(input_nodes,hidden_nodes,output_nodes,0)
    offspring[i] = person[i]
    print('{}\t{}\t{}'.format(i,
                             person[i].who,
                             'UNKNOWN'))
```

2. Calculate fitness.

Run all the individual agents and calculate the fitness (How far the agent can survive). In this case, we just have one agent as one machine to be evaluated simultaneously.

3. Survival of the fittest.

Survival of the fittest describes how birds tend to live longer and reproduce, creating a new generation with evolved genes (the Artificial Neural Network).

```
#Mom
i1 = random.randrange(POP_SIZE) # choose parent
i2 = random.randrange(POP_SIZE) # choose parent
i3 = random.randrange(POP_SIZE) # choose parent

#Tournament
if person[i1].fitness >= person[i2].fitness:
    mom = i1
else:
    mom = i2
if person[i3].fitness >= person[mom].fitness:
    mom = i3

#Dad
i1 = random.randrange(POP_SIZE) # choose parent
i2 = random.randrange(POP_SIZE) # choose parent
i3 = random.randrange(POP_SIZE) # choose parent

#Tournament
if person[i1].fitness >= person[i2].fitness:
    dad = i1
else:
    dad = i2
if person[i3].fitness >= person[dad].fitness:
    dad = i3
```

4. Reproduction

Reproduction to create a new generation of offspring.

```
#Crossover

# Crossover for who
for i in range(hidden_nodes):
    if random.random()< X_BIAS:
        offspring[indiv].who[0,i] = person[mom].who[0][i]
    else:
        offspring[indiv].who[0,i] = person[dad].who[0][i]

# Crossover for wih
for i in range(input_nodes):
    for ii in range(hidden_nodes):
        if random.random()< X_BIAS:
            offspring[indiv].wih[ii,i] = person[mom].wih[ii][i]
        else:
            offspring[indiv].wih[ii,i] = person[dad].wih[ii][i]
```

5. Mutation

Mutation to evolve and enlarge the search space.

```
#Mutation

# Mutation for who
for i in range(hidden_nodes):
    if random.random()< MUT_RATE:
        r = (random.randint(0, 1))%2 *2-1 # create a number either -1 or 1 (sign)
        offspring[indiv].who[0,i] += r*STEP_SIZE

# Mutation for wih
for i in range(input_nodes):
    for ii in range(hidden_nodes):
        if random.random()< MUT_RATE:
            r = (random.randint(0, 1))%2 *2-1 # create a number either -1 or 1 (sign)
            offspring[indiv].wih[ii,i] += r*STEP_SIZE
```

6. Next generation

All parents are replaced by the offsprings.

4.1.4 Learning Transfer

After hours of training and calibration, this is the result that you might get. Otherwise, if you want to save time, you could use the following weights.

Pre-trained Artificial Neural Network weights:

```
self.wih = numpy.array([[-3.29829867,  0.76441159,  2.40556884, -0.27947868],
                        [ 0.3240327,   2.25160679, -1.84486032, -1.9700158 ],
                        [-2.62540542,  2.33445864, -0.46812661, -2.3635345 ],
                        [-0.90241636,  1.99882317, -2.67465566, -1.29619994],
                        [-0.15231266,  2.49082877,  1.08143091,  0.58047555],
                        [ 0.62497593,  1.42985698, -3.91579115, -0.20542114],
                        [ 2.7336978 ,  2.26497664,  1.86146316, -0.69662931]])

self.who = numpy.array([[-4.23668794, -1.10929065,  0.05054322,  0.41018827,  2.70858315, -0.42650511, -1.21117085]])
```

4.2 Hardware: Robot

The robot is a simple clicker. The task is to click a space bar on a keyboard.

Part lists:

- Arduino Nano
- SG90 Micro Servo
- M2 screws and nuts
- 2x 3D printed parts
- Wires

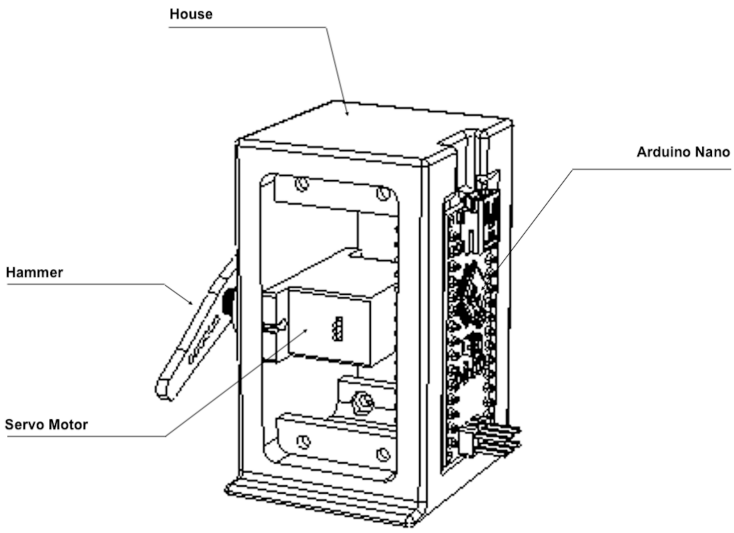
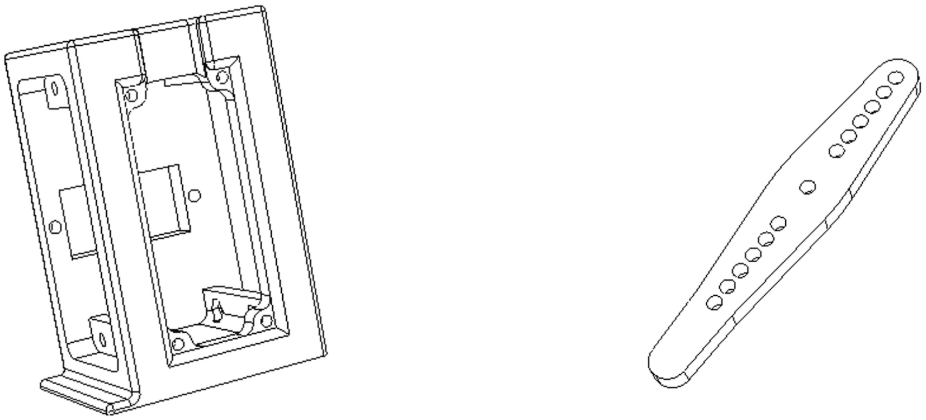


Figure 24: 3D Model.

The actuator used is a servo. To control the servo, a PWM signal is transmitted to the micro servo to locate where it should be. We used a generic development microcontroller board (Arduino Nano) to get the computer's command using a USB interface. The computer gets the inputs from all four inputs from the game and then gives one output value up or down via USB to Arduino Nano. Up is when the robot needs to go up, and down is when the robot needs to click the space bar.

4.3 3D CAD



(a) House.stl

(b) Hummer.stl

Figure 25: 3D printed parts.

4.4 Microcontroller (Arduino Nano)

The arduino Nano board uses AT MEGA 328 P microcontroller, the AT MEGA 328 P has 32 pins. First, a microcontroller takes some types of input. Second, a microcontroller takes a decision based on the software written and lastly the outputs are changed based on the decision made in the second step.All of the steps are done from the program which is written by you and the program runs very quick.

In the AT Mega 328 all the inputs and outputs will be judged based on 0, 5 volts or in between. If you look at the datasheet on the pin confifurations almost all of the 32 pins have multiple functions and that is why there is a long lines of text next to each except for some pins. Those pins are the pins having to do with voltage and ground like VCC, GND, AVCC, AREF and AGND.

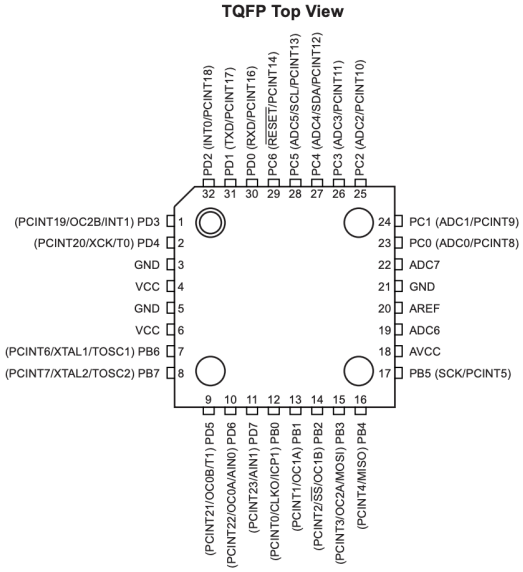


Figure 26: AT Mega 328P Pinout.

Sensors and actuator are electronic components that allow a piece of electronics to interact with the world. We live in an analog world. Monitoring and controlling the real environment is commonly analog in nature. To interface with the environment, Analog to Digital conversation (ADC) and Digital to Analog Conversion (DAC) are required.

An input device is an hardware device able to produce an electrical signal depending on a physic input. The electrical signal changes accordingly to the changes of the input and can be read from a computer. All the sensors can be considered input devices and also a camera, a keyboard, a mouse and so o. Due to presence of Analog to Digital Converter (ADC) and dedicated functions in a microntroller is well suited directly read data from sensors.

An output device is an hardware device able to produce a physical output given a specific electrical signal. The output changes accordingly to the electrical signal in input that can be programmed from a computer. Any kind of electrical actuators can be considered as output devices but also a printer, a speaker, etc. A microcontroller is able to generate the proper electrical signals to control a wide range of output devices.

Working with electronics means dealing with both analog and digital signals, inputs and outputs. Our electronics projects have to interact with the real, analog world in some way, but most of our microprocessors, computers, and logic units are purely digital components.

Similar to the signal we have for input devices, to drive an output device there are different kind o for output signal. In this project we use a servo motor. A servo motor requires PWM (Pulse Width Modulation) signal. Depending on the charracteristic of the PWM signal, the servo rotate in desired angle.

Most microcontrollers provide only digital outputs (logic 0 or 1). Transistors are also much more efficient when acting as on and off switches, rather than producing intermediate outputs. They dissipate a lot of power when neither completely on nor

off. How can we control an output continuously by only switching it on and off? The common method of simulating a continuously variable output is pulse width modulation (PWM): The duty cycle of a square wave is varied to change the average power delivered to a load.

For example, the period of the square wave is 256 units and the duration of the pulse output is 1 from 0, fully off to 256, fully on.

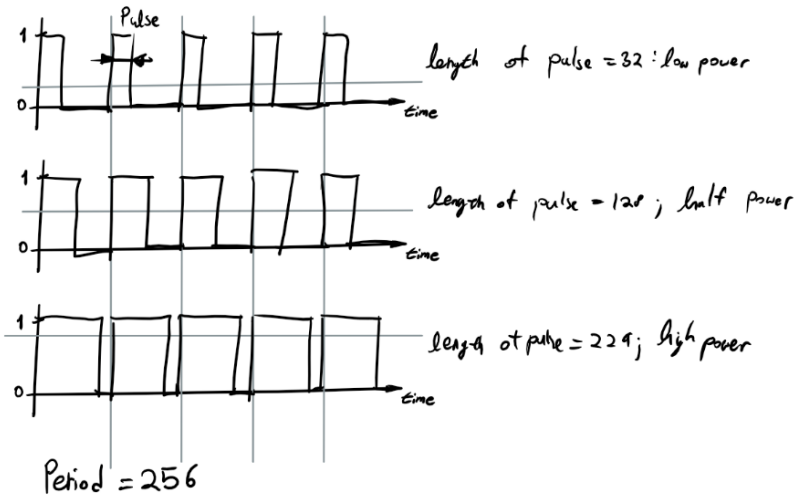


Figure 27: Pulse Width Modulation.

Servos are controlled by a stream of pulses: The position of the servo depends on the length of pulse. For example, 1.0 ms - far left (-45 degree or - 90 degree), 1.5 ms - centered (0 degree) and 2.0 ms - far right (+45 degree or +90 degree).

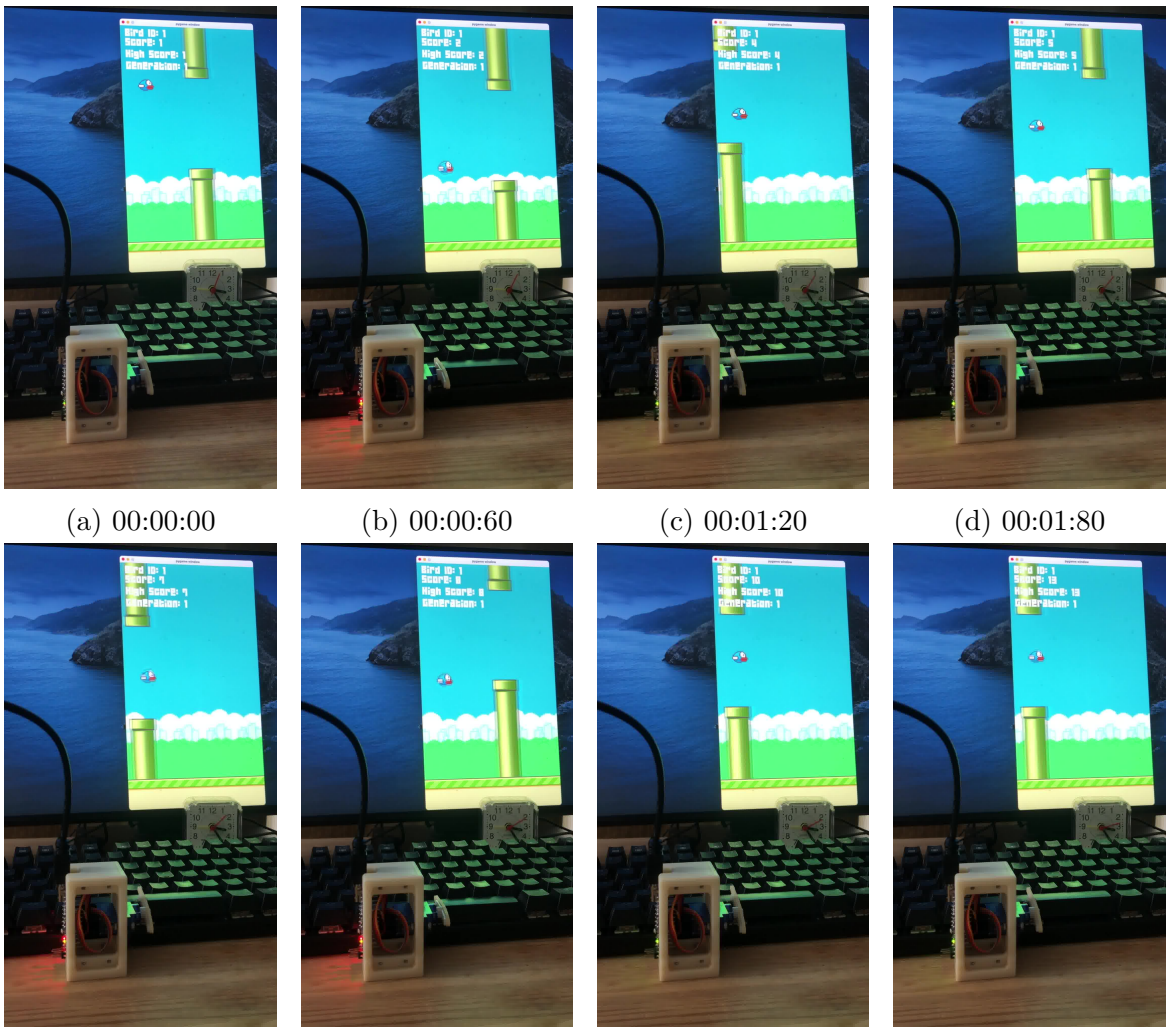
```
1
2  /*
3    modified 17 Apr 2021
4    by Marcello tania
5    This work may be reproduced, modified, distributed,
6    performed, and displayed for any purpose. Copyright is
7    retained and must be preserved. The work is provided
8    ; no warranty is provided, and users accept all
9    liability.
10 */
11
12
13 #include <Servo.h>
14
15 Servo myservo; // create servo object to control a servo
16 const int ledPin = 13; // the pin that the LED is attached to
17 int incomingByte;      // a variable to read incoming serial data into
18
19 void setup() {
20   // initialize serial communication:
21   Serial.begin(38400);
```

```
22   // initialize the LED pin as an output:
23   pinMode(ledPin, OUTPUT);
24   myservo.attach(9); // attaches the servo on pin 9 to the servo object
25 }
26
27 void loop() {
28   // see if there's incoming serial data:
29   if (Serial.available() > 0) {
30     // read the oldest byte in the serial buffer:
31     incomingByte = Serial.read();
32     // if it's a capital H (ASCII 72), turn on the LED:
33     if (incomingByte == 'H') {
34       digitalWrite(ledPin, HIGH);
35       myservo.write(8); // sets the servo position according to the
    scaled value
36       delay(20);
37     }else if (incomingByte == 'L') {
38       digitalWrite(ledPin, LOW);
39       myservo.write(30); // sets the servo position according to the
    scaled value
40       delay(20);
41     }else if (incomingByte == 'R') {
42       digitalWrite(ledPin, LOW);
43       myservo.write(30); // sets the servo position according to the
    scaled value
44       delay(200);
45     }
46
47   }
48   Serial.println(Serial.read());
49 }
```

5 Result and Discussion

Congratulations AI! You play yourself. Well, the comparison between having a small robot and not actually having a robot to play the game might be interesting to be seen during this project. Pure simulation in the software gives us a speedy learning process, less than an hour to be able to play the game. However, if we added a robot, the learning process would take almost a week. What is astonishing is the neuroevolution can adjust all this delay from the computer to the little robot and all the hardware delay without explicitly telling the computer program that we added a robot.

To minimize the training process, I tried to transfer the weights from simulation and then add the robot. Statically I think learning transfer saves me some time than having to train it from scratch. In addition, the number of neurons and layers are still mystery. Regardless one hidden layer or two hidden layer, the computer can still play the game and adjusting the weights between the neurons.



(a) 00:00:00 (b) 00:00:60 (c) 00:01:20 (d) 00:01:80

(e) 00:02:40 (f) 00:03:00 (g) 00:00:00 (h) 00:03:60

Figure 28: Image frame of a machine playing Flappy bird.

6 Conclusion and Future Work

Artificial Neural Network (ANN) is a mathematical model with a lot of knobs to adjust the strength of the signal going to the next neurons. Optimization can be solved by using evolutionary algorithm which is used to optimized the knobs between neurons. Combining the ANN and EA makes an Artificial Intelligent (AI). This paper proofs the implementation of a robot to teach itself how to play a game.

Some inputs parameters of the games stimulate the ANN and then being optimize using EA, giving an output what to do at the very end. After generations the computer learns how to improve to reach the highest score.

As this work has inspired me to work further creating almost anything. I learned that we need to learn more about biology as a roboticist. Study on origin of the organism

where living things are governed and hard coded by the DNA. The rules can be applied to a simulation with help of evolution to create a creature that evolve depending on the fitness we previously defined. As a result, we can generate an automatic hardware design where we can let the computer evolving the DNA of an organism to be a robot that function as we targeted to become. A machine that can design and construct almost anything. Something that can walk, jump, swim and fly or doing almost anything. Previously was about constructing the brain and the future work would be the hardware design.

7 List of Acronyms

8 Appendices

```
/**
 * Evolutionary Algorithms WS 2020/2021
 * Prof. Achim Kehrein
 * Practical Training 1
 * Date : 16 Nov 2020
 * By: Marcello Tania

Find the maximum of
f(x) = 2x^3 -3x^2 -12x + 1 on [-2,3]

**/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


#define POP_SIZE 10 //define population size
#define STEP_SIZE 0.1 // definie step size
#define MAX_GEN 50 // define the maximum generations
float pheno2fitness(float ph);
float findMax(float fit);

float pheno[POP_SIZE]; /* phenotype */
float fitness[POP_SIZE];
int indiv; /* index for individuals */
int parent;  /* index for parent */
int rival;
float offspring_phenotype;
float offspring_fitness;
float best_fitness[MAX_GEN];
float average_fitness[MAX_GEN];
int generation = 0;

int main(void) {

    srand(time(NULL));



    // 5. Choose the best individual
    // Create an initial parent and choose the best individual
    printf("Individual \t Phenotype \t Fitness\n");

    // create first individual
    pheno[0] = (rand() / (float)RAND_MAX) * 5.0 - 2.0;
    fitness[0] = pheno2fitness(pheno[0]);
    printf("\t %d \t\t %6.2f \t %6.2f\n", 0, pheno[0], fitness[0]);

    // initialize statistical analysis
    best_fitness[generation] = fitness[0];
```

```
51      average_fitness[generation] += fitness[0];
52
53
54      // 2. Create an inital population often uniformly randomly over the
        search space [-2,3]
55      for (indiv =1; indiv < POP_SIZE; indiv++) {
56          // 1. Representation of individuals take number in [-2,3] ,
    fitness, use : f(x)
57          // create individuals
58          pheno[indiv] = (rand()/(float) RAND_MAX)* 5.0 -2.0;
59          fitness[indiv] = pheno2fitness(pheno[indiv]);
60          printf("\t %d \t\t %6.2f\t\t %6.2f\n", indiv, pheno[indiv],
    fitness[indiv]);
61
62          // update statistical analysis
63          average_fitness[generation] += fitness[indiv];
64          if (fitness[indiv] > best_fitness[generation]){
65              best_fitness[generation] = fitness[indiv];
66          }
67      }
68
69      average_fitness[generation] = average_fitness[generation] /
    POP_SIZE;
70
71      printf("The average fitness is %6.2f\n", average_fitness[generation
    ]);
72      printf("The best fitness is %6.2f\n", best_fitness[generation]);
73
74
75      // 3. Select parents and create offspring mutiation of traits.
76      // Selecting parents and generating offspring phenotype
77      parent = rand() % POP_SIZE;
78      rival = (rand() % 2) * 2 - 1;                                   //
    creates a number either -1 and 1
79      offspring_phenotype = pheno[parent] + (rival * STEP_SIZE); //
    offspring phenotype by mutation
80      offspring_fitness = pheno2fitness(offspring_phenotype); //
    calculating offspring fitness
81
82      if (offspring_phenotype > 3)
83      {
84          offspring_phenotype = 3 - (offspring_fitness - 3);
85      }
86      if (offspring_phenotype < -2)
87      {
88          offspring_phenotype = -2 - (offspring_fitness + 2);
89      }
90
91      printf("The chosen parent is %d\n", parent);
92      printf("The phenotype of the offspring is %6.2f\n",
    offspring_phenotype);
93      printf("The fitness of the offspring is %6.2f\n", offspring_fitness
```

```
    );

    //4. "Strugle for survival"; selection of next generation
    //Rival vs Offspring

    rival = rand() % POP_SIZE;
    printf("The chosen rival is %d\n", rival);
    printf("The fitness of the rival is %6.2f\n", fitness[rival]);

    if (offspring_fitness > fitness[rival]) // if offspring is fitter
than rival, replace rival
    {
        pheno[rival] = offspring_phenotype;
        fitness[rival] = offspring_fitness;
    }

    // output new population
    printf("Individual \t Phenotype \t Fitness\n");
    for (indiv = 0; indiv < POP_SIZE; indiv++)
    {
        printf("\t %d \t\t %6.2f \t %6.2f\n", indiv, pheno[indiv],
fitness[indiv]);
    }

    // do statistical analysis
    // repeat for fixed number of times
    // keep track of best individual_so_far and average fitness of
population for each generation

    // 6. Repeat to have more generation
    for (generation = 0; generation < MAX_GEN; generation++) {
        printf("------------------------------\n");
        printf("Generation : %d\n", generation);
        // 6.3 Select parents and create offspring mutiation of traits.

        parent = rand() % POP_SIZE;
        rival = (rand() % 2) * 2 - 1;                                  //
creates a number between -1 and 1
        offspring_phenotype = pheno[parent] + (rival * STEP_SIZE); //
calculating offspring phenotype

        if (offspring_phenotype > 3)
        {
            offspring_phenotype = 3 - (offspring_fitness - 3);
        }
        if (offspring_phenotype < -2)
        {
            offspring_phenotype = -2 - (offspring_fitness + 2);
        }

        offspring_fitness = pheno2fitness(offspring_phenotype); //
calculating offspring fitness
```

```
139         /**
140         printf("The chosen parent is %d\n", parent);
141         printf("The phenotype of the offspring is %6.2f\n",
    offspring_phenotype);
142         printf("The fitness of the offspring is %6.2f\n",
    offspring_fitness);
143         **/
144 
145 
146         // 6.4. "Strugle for survival"; selection of next generation
147         //Rival vs Offspring
148 
149         rival = rand() % POP_SIZE;
150         /**
151         printf("The chosen rival is %d\n", rival);
152         printf("The fitness of the rival is %6.2f\n", fitness[rival]);
153         **/
154         if (offspring_fitness > fitness[rival])
155         {
156             pheno[rival] = offspring_phenotype;
157             fitness[rival] = offspring_fitness;
158         }
159 
160         //printf("Individual \t Phenotype \t Fitness\n");
161         for (indiv = 0; indiv < POP_SIZE; indiv++)
162         {
163             fitness[indiv] = pheno2fitness(pheno[indiv]);
164             //printf("\t %d \t\t %6.2f \t %6.2f\n", indiv, pheno[indiv
    ], fitness[indiv]);
165 
166             // find the best fitness
167             average_fitness[generation] += fitness[indiv];
168 
169             //printf("before The best fitness is %6.2f\n", best_fitness
    [generation]);
170 
171 
172             if (fitness[indiv] > best_fitness[generation]){
173                 best_fitness[generation] = fitness[indiv];
174             }
175         }
176 
177 
178         average_fitness[generation] = average_fitness[generation] /
    POP_SIZE;
179         printf("The average fitness is %6.2f\n", average_fitness[
    generation]);
180         printf("The best fitness is %6.2f\n", best_fitness[generation])
    ;
181 
182     }
183 
```

```
184     // 7. Print Generation vs best fitness and average
185      printf("*************************\n");
186     printf("Generation \t Average fitness \t Best fitness\n");
187 
188     for (generation =0; generation < MAX_GEN; generation++) {
189             printf("\t %d \t\t\t %6.2f \t\t %6.2f\n", generation,
    average_fitness[generation], best_fitness[generation]);
190     }
191 
192     return EXIT_SUCCESS;
193 }
194 
195 float pheno2fitness(float ph) {
196     return ((2.0 * ph - 3.0) * ph - 12.0) * ph + 1.0;
197 
198 
199 }
```

References

[1] B. Takács, J. Holaza, J. Števek, and M. Kvasnica, "Export of explicit model predictive control to python," in *2015 20th International Conference on Process Control (PC)*, 2015, pp. 78–83.

[2] A. Sahin, E. Atici, and T. Kumbasar, "Type-2 fuzzified flappy bird control system," in *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016, pp. 1578–1583.

[3] K. A. D. Jong, "Evolutionary computation - a unified approach," 2006.

[4] N. Campbell and J. Reece, "Descent with modification: A darwinian view of life," 2008.

[5] T. Jansen, "Strandbeesten - the new generation," 2022.

[6] M. A. Wainberg, W. C. Drosopoulos, H. Salomon, M. Hsu, G. Borkow, M. Parniak, Z. Gu, Q. Song, J. Manne, S. Islam, G. Castriota, and V. R. Prasad, "Enhanced fidelity of 3tc-selected mutant hiv-1 reverse transcriptase," 1996.

[7] P. Braspenning, F. Thuijsman, and A. Weijters, "Artificial neural networks: An introduction to ann theory and," p. 15, 1995.

[8] R. B. S. Agatonovic-Kustrin *, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," 1999.

[9] J. Henseler, "Back propagation," 2005, braspenning, P.J., Thuijsman, F., Weijters, A.J.M.M. (eds) Artificial Neural Networks. Lecture Notes in Computer Science, vol 931.