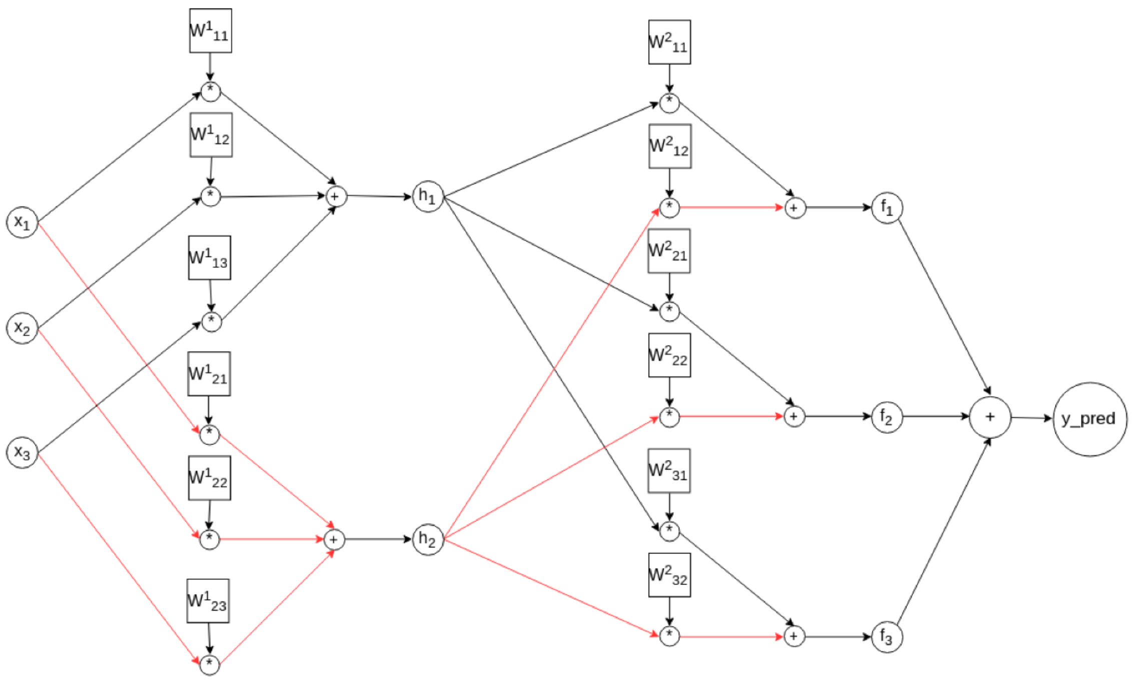
CS 451/551 Introduction to Artificial Intelligence

Assignment – 2: Evolutionary Process Optimizer

Created by Anıl Doğru & Mehmet Onur Keskin for help → {anil.dogru, onur.keskin}@ozu.edu.tr

1. Definition

In this assignment, you will implement a genetic algorithm to optimize the following process. The process involves performing some series of mathematical operations on the given diagram. An array of real numbers **x** = [x1, x2, x3] is given to some parameters called **weights** denoted as W, and after performing those operations illustrated in the figure, the system will produce a real-valued output, called **y**$_{\textbf{predict}}$. Note that the color of the arrows has no functionality. It is to make the process visually traceable.



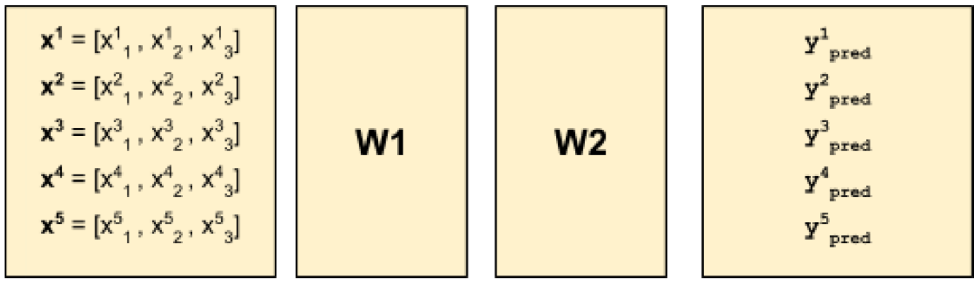
Sample calculations are as follows:

$$h_1 = (w_{11}^1 * x_1) + (w_{12}^1 * x_2) + (w_{13}^1 * x_3)$$

$$f_1 = (w_{11}^2 * h_1) + (w_{12}^2 * h_2)$$

$$y_{pred} = f_1 + f_2 + f_3$$

If you notice, the process can easily be converted into a linear algebra representation – matrix multiplications (*highly recommended to reduce complexity)*. In the following example, we have an input having N training instances where N equals 5. The input **X** is a [5 x 3] matrix having five training instances, and the calculated output **y** is a [5 x 1] vector. **y** is calculated by performing the multiplications mentioned above and summations with corresponding weights.


$\mathbf{x}^1 = [x^1_1, x^1_2, x^1_3]$
$\mathbf{x}^2 = [x^2_1, x^2_2, x^2_3]$
$\mathbf{x}^3 = [x^3_1, x^3_2, x^3_3]$
$\mathbf{x}^4 = [x^4_1, x^4_2, x^4_3]$
$\mathbf{x}^5 = [x^5_1, x^5_2, x^5_3]$
W1
W2
y^1_{pred}
y^2_{pred}
y^3_{pred}
y^4_{pred}
y^5_{pred}

2. Genetic Algorithm as optimizer:

In Computer Science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. John Holland introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution; his student David E. Goldberg further extended GA in 1989.

For more information:

- Wiki
- Genetic Algorithms - Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand. An excellent introduction to GA by John Holland and with an application to the Prisoner's Dilemma
- An online interactive Genetic Algorithm tutorial for a reader to practice or learn how a GA works: Learn step by step or watch global convergence in batch, change the population size, crossover rates/bounds, mutation rates/bounds and selection mechanisms, and add constraints.
- Genetic Algorithms in Python Tutorial with the intuition behind GAs and Python implementation.

In this assignment, you will find the most convenient weight values (12 weights) by applying the genetic algorithm. A gene represents the value of each weight. A chromosome will consist of 12 genes corresponding to the value of weights. Your algorithm will start with a population - a set of chromosomes - as candidate solutions for your problem.

You should implement the five phases of a genetic algorithm:

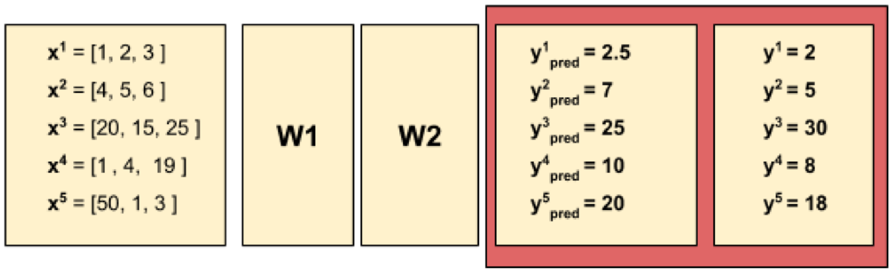
1. Initial Population
2. Fitness Function
3. Selection
4. Crossover
5. Mutation

The “survival of the fittest” principle must be embedded by intuition. However, in this study, **the gene *Population size* stays the same.** When the weakest individual dies, it is replaced. For that reason, it is normal to have similar individuals in your population as they demonstrate higher fitness.

- Hierarchy: *Genes → Individuals (Chromosomes) → Population*

The system has only linear components; output must also be a linear combination of inputs. **You will calculate your real output/target as the *mean of rows.*** See the example:

- $y^1 = mean(x^1) = \frac{1+2+3}{3} = 2,\ where\ x_{11} = 1,\ x_{12} = 2,\ x_{13} = 3$



Given the inputs and an individual of the population (gene string of weights, which is not shown in the figure), the system produced close but inaccurate results. *It seems like it is on the excellent track of fitness.*

Fitness measure for evaluation
Our fitness measure for the individuals of the population will be the sum of root squared error, which is

$$\sum_{i}^{Row} \sqrt{\left(y^i - y^i_{pred}\right)^2}$$

We strongly recommend & ask you to read about the algorithm and the problem before starting to implement your assignment. Provided class files should be investigated before mailing to TA. TA will NOT help to code!

3. Implementation Details

- The input matrix should be [Nx3] where N ≥ 16, and the population matrix should have the size of [Mx12] where M ≥ 20
    - Benchmark: A 8-year-old laptop with Intel® Core™ i5-4210U can easily handle 100 rows of input with a population having 200 individuals.
- Input matrix **X** should have actual values in [-5,5].
- Your weights/genes should have values in [-1,1]. Their initial values and mutated genes will be generated from this range.
- The algorithm should randomize floating numbers (not **integers)**. Random generator's distribution is up to you: it can be a Gaussian or uniform distribution. The mean of the generation ranges may not be zero; given that range, you are free to play.
- You can use ***data_generator.py*** to generate random data. You can edit Size, Range, File Name, etc., or implement your data generator.
- You can use NumPy, Pandas, Matplotlib and Seaborn Python libraries.
- The stopping condition for the evolution may be an iteration limit like 2000 or a condition such as "until fittest individual provides fitness error at most 0.05". However, your algorithm must progress in the end as you will plot its fitness progress over iterations for the genes (see example plot).
- There should be a line to print which generation is being created (number) and each of the population members' fitness levels (see example), not necessarily after each iteration. It may be after each T number of iterations.
- You may have as much as helper methods or classes you like. Clean coding is appreciated.
- It is highly recommended to improvise and research for the related methods to reach faster convergence ability and escape from sub-optimal solutions to be used in Mutation and Crossover.
- You cannot use linear algebra solvers or implement closed-form solutions like $\hat{\beta} = (X^T X)^{-1} X^T y$.
- Please add comments to your methods and pick descriptive names for the variables.
- You may be inspired by the source codes you found, but plagiarizing is strictly forbidden. This algorithm and this problem are well-studied in the literature and online resources, and there are many different coding examples online. We have almost all of them, and please do NOT try to use them directly. If you copy online resources directly, you will get 0 points.

Note: If you are not comfortable writing code in Python, you may follow this tutorial to improve your Python programming skills. Even if you do not know anything about Python, you only need 2-3 days to learn Python from scratch to complete this assignment. If you have any questions about the implementation, please do not hesitate to e-mail the assistant.

4. Criterion

- **Report:** You need to write a report to explain your design in detail. One page code explanation is not a report, and such reports will be ignored. Please follow the rules of technical report writing like the example report:
    - Sample Sections: Introduction, Algorithms, Implementation Details, Results, Conclusion.
    - Figures are significant to explain, but their legend, label or title is insufficient to show your work. They are helpers, and they require some text.
    - Please not copy/paste your code snippets. On the other hand, their formal/non-formal pseudocode with explanation would be perfect.
    - ***There must be a graph showing your individuals' decreasing fitness error with comments on it.***
    - "It was not asked, so I did not write" is the approach of freshmen. Please feel free to add and improvise any related content to show your work.
    - If you do not write your code from scratch, please include your references at the end of the report.
- **Runtime:** Please do NOT include any third-party library in your project (except NumPy, Pandas, Matplotlib, Seaborn) because you do not need any extra library. Including different third-party libraries may cause a problem for running your code in different local machines (*e.g.* dependencies etc.), for any dependency on a runtime that the assistant can solve -15 points penalty, for any dependency on a runtime that the assistant cannot solve -60 points penalty.
- **Compilation:** Please make sure that your code is compiled correctly. Not compiling code (*e.g.,* any error etc.) -60 points penalty.
- **Submission:** All project files (*.PY) and report file (.PDF) should be zipped (.ZIP) together, and the filename of the ZIP file should be in the format of "NAME_SURNAME_ID_hw2.zip". Please follow this structure in your submission. Not following -10 points penalty.
- **Group effort:** You may discuss the algorithms and so forth with your friends, but this is individual work, so you must submit your original work. **In any circumstances of plagiarism, first, you will fail the course, and the necessary actions will be taken immediately.**

Deadline: 17 April 2022 – 23.55 via LMS. **(Strict!)**

Grading Criteria:

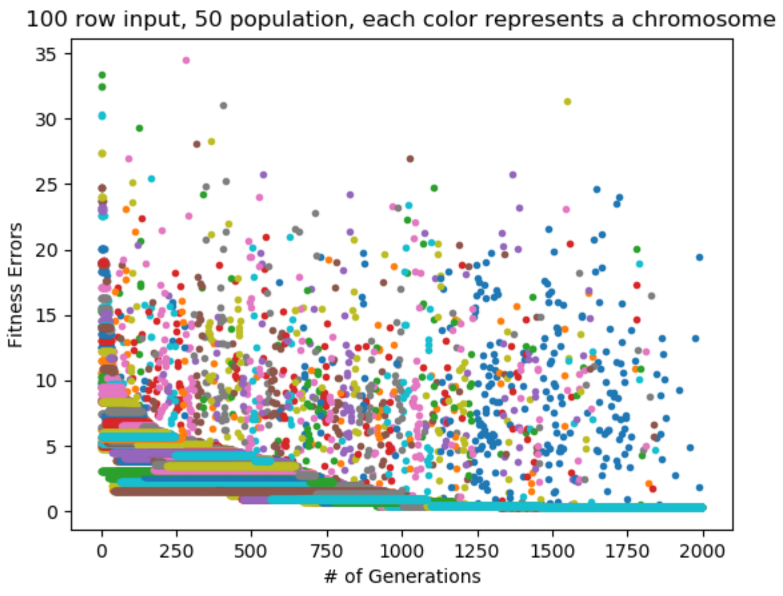
- Evolve: 15
- Crossover: 15
- Mutation: 15
- Tournament: 15
- Report: 40

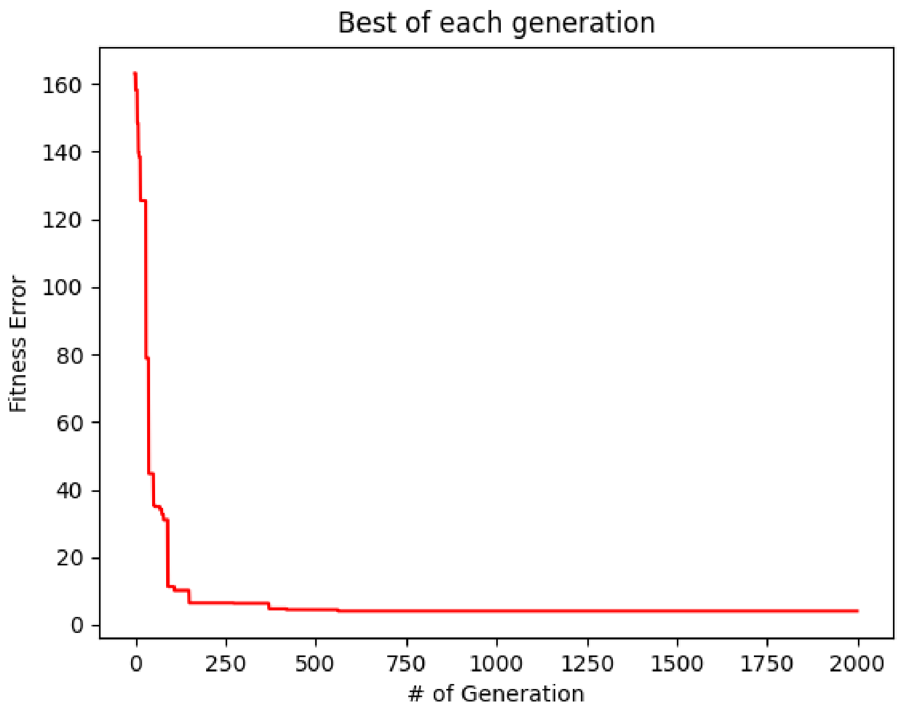
5. Example:

- Input matrix **X** is [100x3] so N = 100 and the matrix of weights, the weights population is [16x12] because M = 16. Note that M < 20, just for this example, prevent it from occupying too much space.
- The limit of iterations is set to 2000 and T = 200. So, in every 200 generations, it prints out the errors of chromosomes.

```
Generation: 1

Fitness errors of chromosomes

[163.266, 190.807, 240.335, 285.231, 304.826, 337.116, 442.085, 443.375, 507.648, 529.060, 549.116,
568.629, 646.351, 657.836, 762.566, 776.621]

Generation: 200

Fitness errors of chromosomes

[6.445, 6.445, 6.445, 6.445, 6.445, 6.445, 6.445, 6.445, 6.445, 6.445, 6.445, 6.445, 73.876, 73.876,
226.182, 300.795]

Generation: 400

Fitness errors of chromosomes

[4.734, 4.734, 4.734, 4.734, 4.734, 4.734, 4.734, 4.734, 4.734, 4.734, 4.734, 4.734, 448.371, 448.371,
448.371, 579.395]

Generation: 600

Fitness errors of chromosomes

[4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042,
71.854]

Generation: 800

Fitness errors of chromosomes

[4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 143.399, 143.399,
143.399, 488.248]

Generation: 1000

Fitness errors of chromosomes

[4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.421, 4.421,
89.781]

Generation: 1200
```

```
Fitness errors of chromosomes

[4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 4.042, 41.958, 66.191, 245.744, 381.840,
458.258, 589.312]

Generation: 1400

Fitness errors of chromosomes

[4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035,
337.568]

Generation: 1600

Fitness errors of chromosomes

[4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 36.499, 53.178, 77.976, 203.342,
203.342, 437.975]

Generation: 1800

Fitness errors of chromosomes

[4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 19.570, 19.570, 20.023, 20.023, 24.743, 40.857, 46.539,
187.653, 187.653, 259.263]

Generation: 2000

Fitness errors of chromosomes

[4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 4.035, 228.080, 228.080,
228.080, 499.162]
```




Best of each generation
Fitness Error
# of Generation
0
20
40
60
80
100
120
140
160
0
250
500
750
1000
1250
1500
1750
2000