# Analysis

## 1. Neural Network

I decided to start with the second part of the assignment, which is the Neural Network Optimization because I think this strategy will give me the advantage of tuning some of the hyperparameters in the Randomized Optimization algorithms, such as the population size in the Genetic algorithm. By doing this, I can use the tuned hyperparameters in these algorithms while applying them to the problems in the first part of the assignment.
I used Randomized Hill Climbing, Simulated Annealing and Genetic algorithms in the Neural Network model created in Assignment 1 with the same structure, the only change is the optimization algorithm, I used the **Heart Dataset** in this problem. After we fit the model into the data using the optimization algorithms, the learning curves are plotted and the model performance is recorded for further optimization and hyperparameters tuning, and that concludes the first step.
The second step, I used GreadSearchCV to get the best weights for each hyperparameter in the model, examples of the tuned hyperparameters are population size and mutation probability in the Genetic algorithm, clip max and step size in the simulated annealing algorithm. Results and analysis are discussed in section 1.1 and 1.2.

### 1.1 Results

#### A. Simulated Annealing

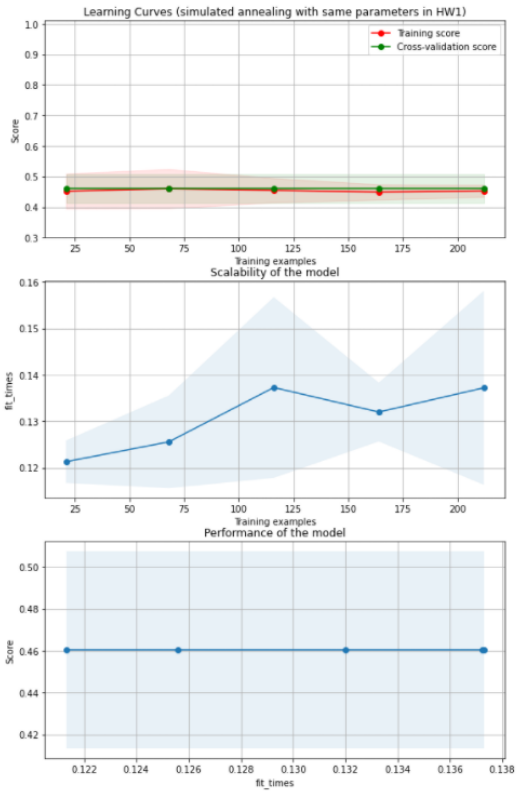
In Simulated Annealing we use the so-called 'Temperature' factor in deciding whether we are going to take a step like a random walk or a step like hill climbing towards the maxima. Below the formula for Exponential Decay which I chose to use in my problems. The advantage of the exponential decay is that it gives us the opportunity of decreasing T slowly and gives the system a chance to explore the current temperature before we start cool it out.

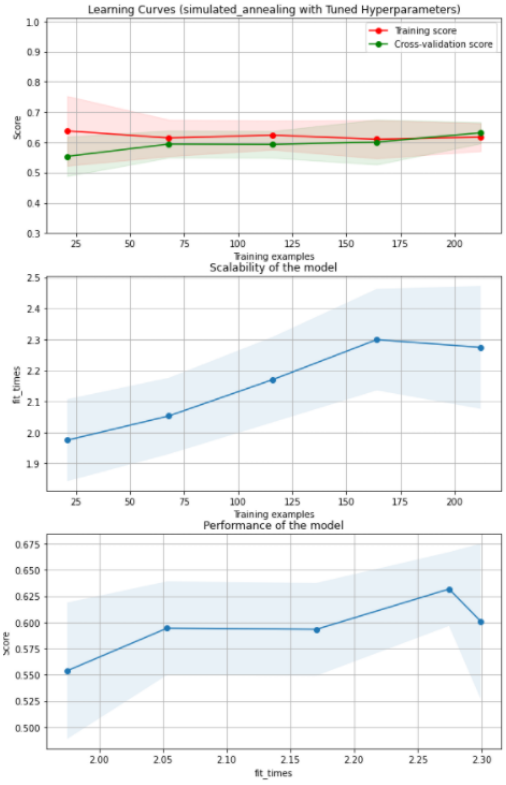
$$T(t) = \max(T_0 e^{-rt}, T_{min})$$

where:

- $T_0$ is the initial temperature (at time t = 0);
- $r$ is the rate of exponential decay; and
- $T_{min}$ is the minimum temperature value.

The training and validation learning curves for Simulated Annealing algorithm with default parameters and same network structure from Assignment 1 compared to the training and validation learning curves after hyperparameters tuning are shown below




On the left, the learning curves for the Simulated Annealing model with the same parameters used in Assignment 1. On the right, the learning curves for the Simulated Annealing model with tuned hyperparameters. We can observe different scores and fit times based on different parameters used in each model. The two tables below highlight the scores and hyperparameters used in each model, respectively.

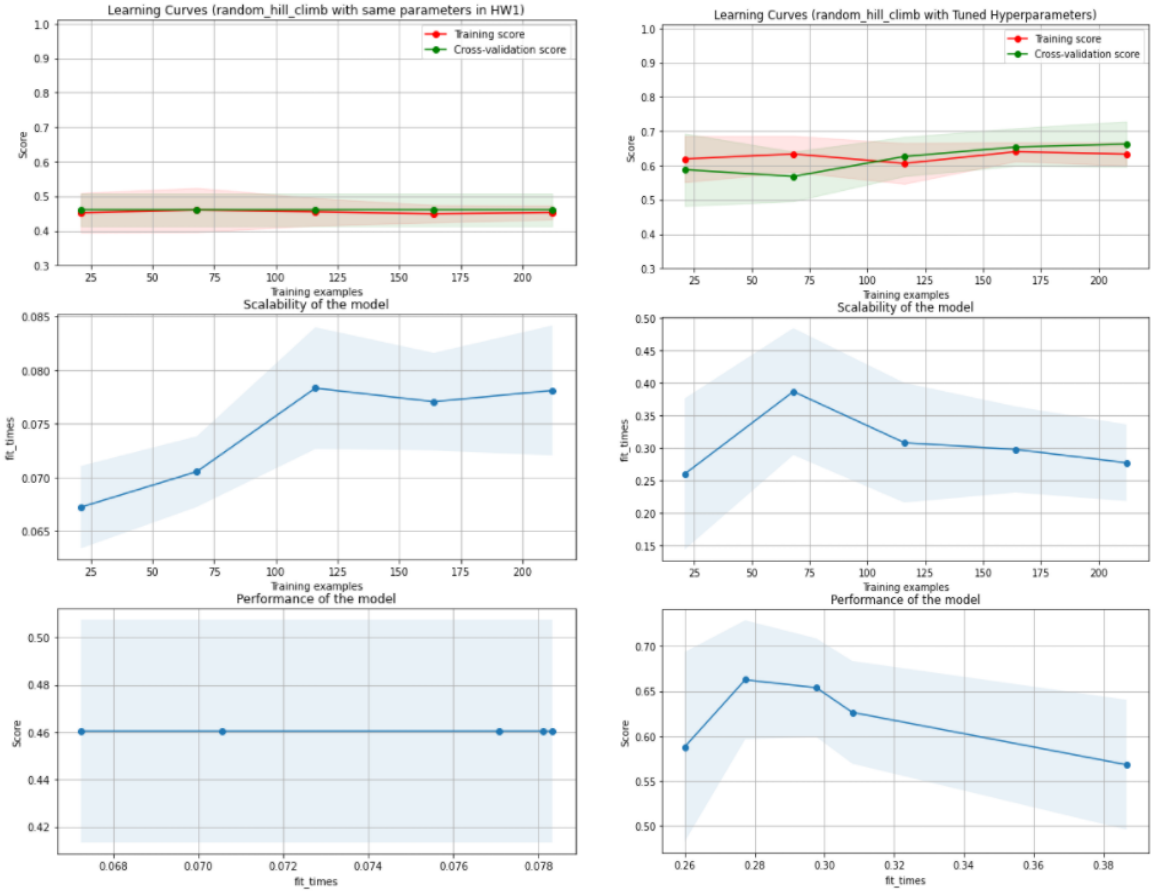
| **Learning Element per 200 Training Examples** | **Simulated Annealing with Same Hyperparameters in Assignment 1** | **Simulated Annealing with Tuned Hyperparameters** | **Difference Green = Positive Red = Negative** |
|---|---|---|---|
| Training Score | 45% | 61% | 16% |
| Validation Score | 45% | 64% | 19% |
| Fit Time (seconds) | 0.138 | 2.29 | 1660% |
| Fit Time for Best Score (seconds) | 0.121 | 2.275 | 1880% |

| **Parameter** | **Simulated Annealing with Same Hyperparameters in Assignment 1** | **Simulated Annealing with Tuned Hyperparameters** |
|---|---|---|
| Hidden_nodes | 8 | 32 |
| Activation | Sigmoid | Sigmoid |
| Max_iters | 100 | 1000 |
| Learning_rate | 0.01 | 0.1 |
| Clip_max | 5 | 2 |
| Max_attempts | 10 | 10 |

## B. Random Hill Climbing

In standard Hill Climbing algorithm, we start with a random point x that has a particular f(x) value, then we move around in the neighborhood around that x point and see where we can go to improve the function value. This approach has a high possibility to stuck in a local maxima region instead of reaching the global maxima. The Random Hill Climbing algorithm will give us a way to deal with the fact that Hill Climbing can get stuck in local maxima. What Random Hill Climbing is going to do is once the local maxima is reached, we will start the whole thing again from some other randomly chosen x until we reach the global maxima.

The training and validation learning curves for Random Hill Climbing algorithm with default parameters and same network structure from Assignment 1 compared to the training and validation learning curves after hyperparameters tuning are shown below



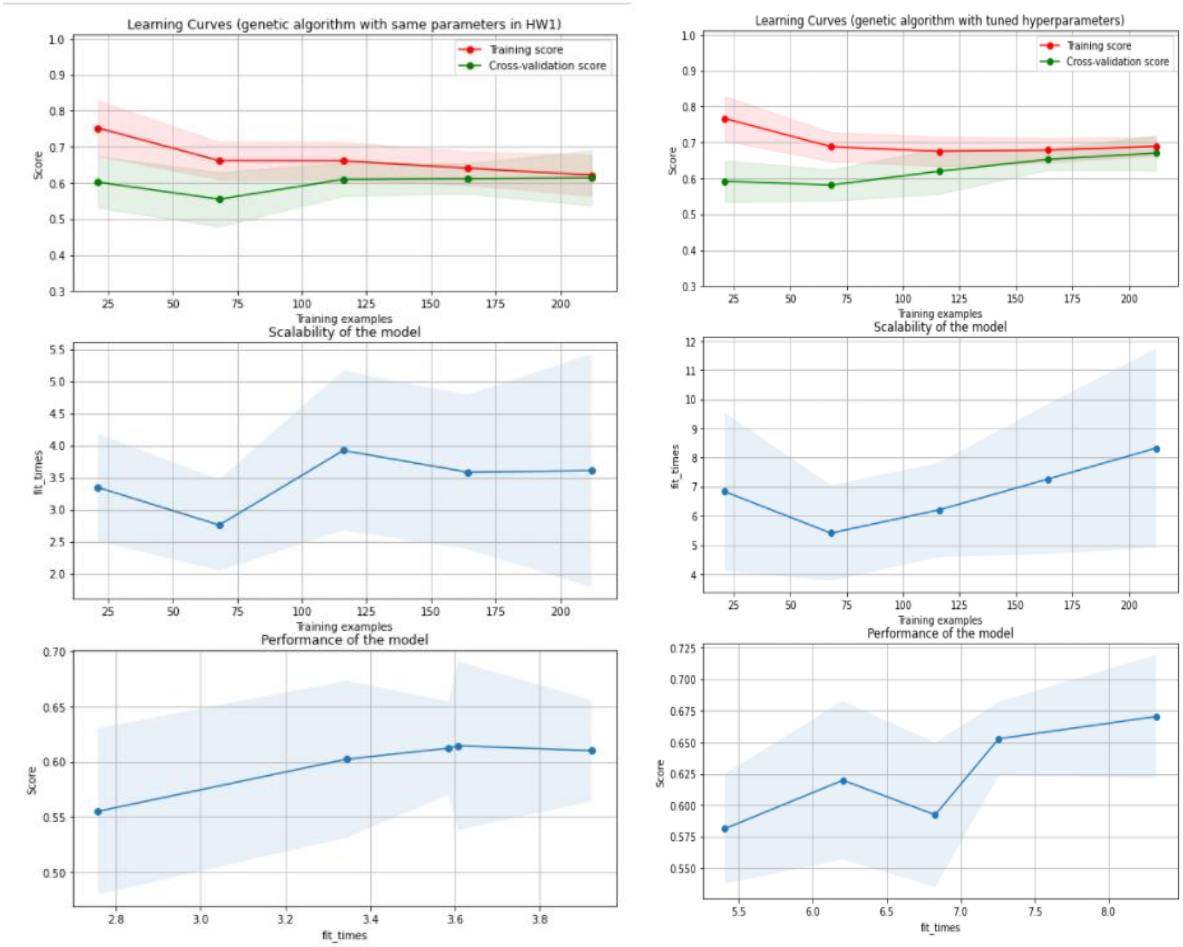

On the left, the learning curves for the Random Hill Climbing model with the same parameters used in Assignment 1. On the right, the learning curves for the Random Hill Climbing model with tuned hyperparameters. We can observe different scores and fit times based on different parameters used in each model. The two tables below highlight the scores and hyperparameters used in each model, respectively

| Learning Element per 200 Training Examples | Random Hill Climbing with Same Hyperparameters in Assignment 1 | Random Hill Climbing with Tuned Hyperparameters | Difference Green = Positive Red = Negative |
|---|---|---|---|
| Training Score | 45% | 63% | 18% |
| Validation Score | 47% | 68% | 21% |
| Fit Time (seconds) | 0.078 | 0.275 | 350% |
| Fit Time for Best Score (seconds) | 0.068 | 0.278 | 405% |

| Parameter | Random Hill Climbing with Same Hyperparameters in Assignment 1 | Random Hill Climbing with Tuned Hyperparameters |
|---|---|---|
| Hidden_nodes | 8 | 2 |
| Activation | Sigmoid | Sigmoid |
| Max_iters | 100 | 1000 |
| Learning_rate | 0.01 | 2 |
| Clip_max | 5 | 5 |
| Max_attempts | 10 | 10 |

## C. Genetic Algorithm

The Genetic algorithm is an algorithm designed for solving optimization problems based on natural selection. It consistently modifies a population of individual solutions. At each step, it selects individuals randomly from the current population and call them parents and uses them to produce children for the next generation. Over successive generations, which is iterations for improvement by the way, the population converges towards the optimal solution. One of the main properties that differentiate Genetic algorithm from another optimization algorithms is Crossover property, where we combine two parents to form children for the next generation. The training and validation learning curves for the Genetic algorithm with default parameters and same network structure from Assignment 1 compared to the training and validation learning curves after hyperparameters tuning are shown below




On the left, the learning curves for the Genetic algorithm model with the same parameters used in Assignment 1. On the right, the learning curves for the Genetic algorithm model with tuned hyperparameters. We can observe different scores and fit times based on different parameters used in each model. The two tables below highlight the scores and hyperparameters used in each model, respectively

| **Learning Element per 200 Training Examples** | **Genetic Algorithm with Same Hyperparameters in Assignment 1** | **Genetic Algorithm with Tuned Hyperparameters** | **Difference Green = Positive Red = Negative** |
|---|---|---|---|
| Training Score | 62% | 71% | 8% |
| Validation Score | 61% | 70% | 7% |
| Fit Time (seconds) | 3.6 | 8.2 | 225% |
| Fit Time for Best Score (seconds) | 3.6 | 8.7 | 240% |

| **Parameter** | **Genetic Algorithm with Same Hyperparameters in Assignment 1** | **Genetic Algorithm with Tuned Hyperparameters** |
|---|---|---|
| Hidden_nodes | 8 | 8 |
| Activation | Sigmoid | Sigmoid |

| | | |
|---|---|---|
| Max_iters | 100 | 1000 |
| Learning_rate | 0.01 | 0.01 |
| Clip_max | 5 | 5 |
| Max_attempts | 10 | 10 |
| Pop_size | 200 | 400 |
| Mutation_prob | 0.1 | 0.1 |

### 1.2 Analysis

I will start by comparing the results of all the Neural Networks using optimization algorithms to the Neural Network implemented in Assignment 1 where I used the optimizer 'Adam' derived from Adaptive moment estimation which is an extension to stochastic gradient descent. None of the optimization algorithm outperformed Adam optimizer in Neural Network problem, where Adam achieved 86% accuracy on the Heart Dataset, while Genetic algorithm(best optimization algorithm performer in the Neural Network model on the same dataset) achieved 68% accuracy on the same dataset. Practical experiments proved to me that gradient descent optimization are the best in converging to the global maxima.

Back to the random optimization algorithms, the below table highlights the analysis and observations of the behavior of each optimization algorithm.

| Simulated Annealing | Random Hill Climbing | Genetic Algorithm |
|---|---|---|
| • Due to slowly decreasing the temperature, the algorithm needed too much tuning to be able to give us acceptable accuracy such as increasing the iterations from 100 to 1000 and increasing the hidden nodes from 8 to 32.<br>• Due to the elevated parameter weights, it took longer time to fit.<br>• Test accuracy score = 68% | • It has the ability to converge faster than the other 2 algorithms before and after parameters tuning due to using big random step sizes equals 2 in this problem, though using 1000 iterations.<br>• Test accuracy = 68% | • The best performer among the 3 algorithms with test accuracy = 70%<br>• It performs well when increasing the population size.<br>• It didn't require many parameters tuning to better perform.<br>• Due to the fact it is using the Crossover rule, it always finds the optimum solution. |

## 2. Optimization Problems

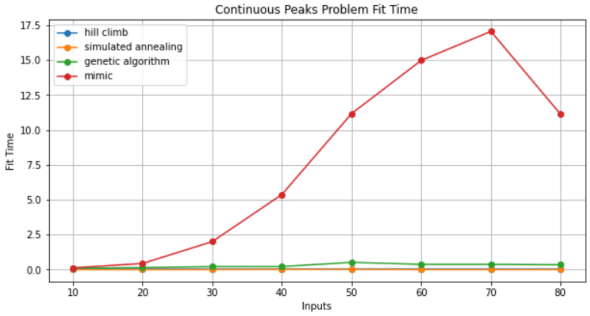
I chose 3 pre-defined optimization problems to evaluate the strengths and behavior of Random Hill Climbing, Simulated Annealing, MIMIC and Genetic algorithms. The three problems are Continuous Peaks, Knapsack and Flip flop.

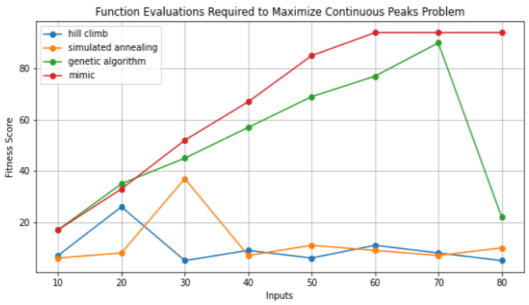
### 2.1 MIMIC Algorithm

I wanted to talk about the MIMIC algorithm before we start exploring the results of all the optimization algorithms on the three problems we defined above.
MIMIC is Mutual Information Maximizing Input Clustering. It achieves maxima by estimating probability densities. MIMIC use the structure of the solution space as guidance for randomized search. MIMIC communicates information about the cost function obtained from one iteration to next iterations of the search.

### 2.2 Continuous Peaks Problem

This problem contains too many local maxima. Below are the time and performance curve for each algorithm. Each run was infinite iterations. MIMIC had the best performance score = 88 with parameters population size = 400, mutation probability = 0.4. However, MIMIC had the worst fit time = 11 seconds for input = 80

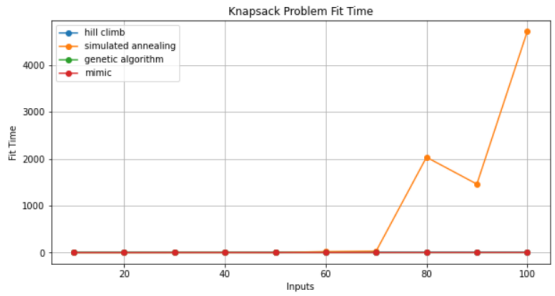


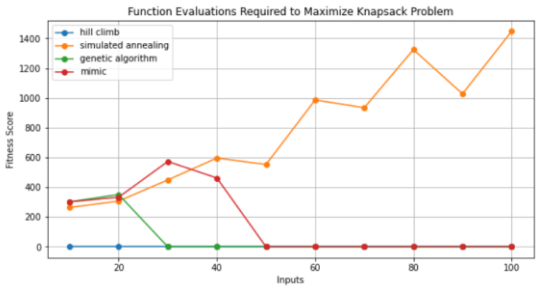




On the left, the fit time curve with respect to the input size. On the right, the fitness score curve with respect to the input size.

### 2.3 Knapsack Problem

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value. The goal is to determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. Below are the time and performance curve for each algorithm. Each run was infinite iterations. Simulated Annealing had the best performance score = 1450 with parameter Geometric Decay, initial temp = 1, decay = 0.99, min_temp = 0.001. However, Simulated Annealing had the worst fit time = 4800 seconds for input = 100

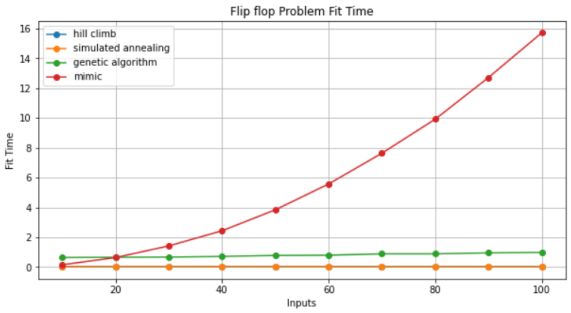


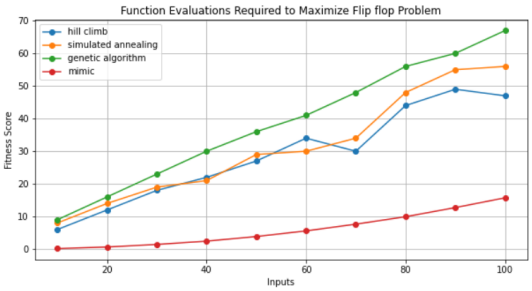




On the left, the fit time curve with respect to the input size. On the right, the fitness score curve with respect to the input size.

## 2.4 Flip-Flop Problem

The Flip-Flop problem counts the number of times of bit alternation in a bit string. Below are the time and performance curve for each algorithm. Each run was infinite iterations. Genetic algorithm had the best performance score = 68 with parameters population size = 200, mutation probability = 0.6. The fit time for Genetic algorithm was good = 1 second for 100 input.




On the left, the fit time curve with respect to the input size. On the right, the fitness score curve with respect to the input size.

## 2.5 Analysis and Conclusion

Testing the 4 optimization algorithms, MIMIC, Genetic, Simulated Annealing and Random Hill Climbing on different optimization problems gives us insights on how each algorithm will behave based on the complexity of the problem. Also, there is a big tradeoff between tuning the algorithm parameter and the computation complexity, for instance, the MIMIC algorithm tends to have high computational cost when we tune the population size. The below table highlights the advantage and disadvantage of each algorithm

| Algorithm | Advantages | Disadvantages |
|---|---|---|
| Simulated Annealing | • Fast | • Needs parameter tuning |
| Random Hill Climbing | • Fast<br>• No parameter tuning needed | • Doesn't converge in complicated problems |

| | | |
|---|---|---|
| MIMIC | • Very good in complicated problems | • Too slow<br>• Needs parameter tuning |
| Genetic | • Performs well in all kind of problems<br>• Fast | • Needs parameter tuning |