Assignment 2: Randomized Optimization

CS7641 Machine Learning, 2019 Fall

Hui Xia (hxia40)
903459648
Georgia Institute of Technology

Part 1. Algorithm optimization over different problems

1. Introduction

An 'engineering' area about machine learning is to find the most suitable parameters for a given set of data. This process is happens in machine learning algorithm, and is usually referred as an optimization problem. In this part of the report, I will investigate the performance of four algorithms, i.e. randomized hill climbing (RHC), simulated annealing (SA), genetic algorithm (GA), and Mutual-Information-Maximizing Input Clustering (MIMIC) against the three randomize optimization problems. Both problems and algorithms are adopted using mlrose [1]. As per required by the assignment, the three problems should highlight the advantages of GA, SA, and MIMIC, respectively. In this part of the report, I will firstly find optimal parameters for each algorithm against each of the problem, respectively, then compare the algorithms, and discuss their pros and cons.

2. Experiment design

2.1. Parameter optimization

To make fair comparisons among all algorithms, I will firstly configure their parameters in practical range, to ensure the performance of each algorithms as high as possible. As there are multiple parameters (e.g. `max_iter`) for each of the algorithms that controls how the algorithm performs, it is not realistic to scan over all combinations of them. Thus, I listed "typical" parameters for each of the algorithms in **Table 1**. For the parameter optimization, while keeping all the other parameters at default, for each of the problem (i.e. the Knapsack problem, or continuous-peaks problem), one of these parameter listed in **Table 1** is changed, and the resulted algorithm performance is plotted against the changing parameter. For each parameter, 50 different values are tested to find the optimal setting. The value of the parameter that grant the maximum fitness score (which will be defined and discussed in details in **Section 2.3**) will be used as the optimal.

Table 1. Algorithm and default parameters for optimization study.

Algorithm	**Hyperparameter**				
	max_attempt	*max_iteration*	*pop_size*	*mutation_prob*	*keep_pct*
RHC	*10*	*10*			
SA	*10*	*10*			
GA	*10*	*10*	*0.1*	*0.1*	
MIMIC	*10*	*10*	*0.1*		*0.1*

2.2. Problems

As per target of this report, I will compare four algorithms using three different problems: I will highlight the advantage of SA using the traveling salesman problem (TSP), highlight GA sing the continuous peaks problem (Cpeaks), and highlight MIMIC using the Knapsack problem (KP). TSP is a NP-hard optimization problem, which is defined by to find the shortest route to travel through all of n "cities" on the map, starting and ending on the same city [1]. The continuous peaks problem is derived from 4-peaks problem [2-3]. In the Cpeaks problem, an array which is made by 0s and 1s is given, and a score is calculated as the longest contiguous bits of 0 or 1. Additionally, a reward is given when there are greater than T (Value of T is defined by user. In our experiment, T = 15% of the total length of the array) contiguous bits set to 0, and greater than T contiguous bits set to 1. Finally, KP is an

optimization problem which defined as, given a set of n items, where item *i* has known weight w_i and known value v_i; and maximum knapsack capacity, W, the KP problem strive to fit items that has the highest total value into an sack. While the total weight of the items cannot exceed the sack capacity W [1].

Among these three problems, the fitness function of the problems Cpeaks and KP target to obtain highest fitness score. However, the fitness function of TSP target to obtain lowest fitness score. To facilitate comparison, in this report, we plot the fitness score of TSP problem as the multiplicative inverse of the actual fitness core.

2.3. Evaluation of algorithm

Using the optimal parameters, the fitness score and the wall clock time for each of the optimization algorithm on each of the problems are plotted. For each problem, the algorithm with the highest fitness score will be considered as the most suitable algorithm. If more than one algorithms have similar (i.e. difference < 5%) fitness score, the algorithm that consume less time will be considered superior in performance. To highlight the randomness of the algorithm, the time and fitness score are evaluated 10 times, and the average and standard deviation of the 10 fitness score/time are presented.

It is worth mentioning that the four algorithms can be categorized into two groups: RHC and SA are "exploitation" algorithms (As the `restarts` is set to 0 for RHC. Although, SA defined a "temperature" function, which decays over time, that let the algorithm perform some explorations), which majorly focus on finding the best decision using given current-available information. On the other hand, GA and MIMIC are "exploitation and exploration" algorithms, which not only exploit given information, but also defined a chance (by `mutation_prob` and `keep_pct` for GA and MIMIC, respectively) to explore potentially worse options. Thus, RHC and SA can quickly perform many iterations of optimization in a short period of time, while GA and MIMIC are relatively computationally expensive. To make fair comparisons among algorithms, in the parameter optimization process (discussed in **Section 2.1**), I will try to use same parameters for RHC and SA, and try to use same parameters for GA and MIMIC.

3. Results and Discussion

3.1. Parameter optimization

As discussed in **Section 2.1**, while keeping all the other parameters at default, one of the parameters listed in **Table 1** is changed and plotted against the changing parameter. **Figure 1** demonstrate the plotted curve for the Cpeaks problem. For algorithms RHC and SA, the fitness score for both `max_attempt` and `max_iteration` plateau when their respective value is equal to or larger than 10. Additionally, `max_attempt` and `mutation_prob/keep_pct` do not affect the fitness score for GA and MMIC algorithms. For GA, `max_iteration` also plateaus when its value is equal to or larger than 10. Also, `pop_size` for both GA and MIMIC plateau when their respective value is equal to or larger than 50 and 100 respectively. As discussed in **Section 2.3**, I will use same parameters for GA and MIMIC, and let both GA and MIMIC use the `pop_size` value of 100.

Similar parameter optimization process was also applied to the problems TSP and KP. Due to the length limitations, I will not describe them in details. The optimized parameters can be found in **Table 2**.

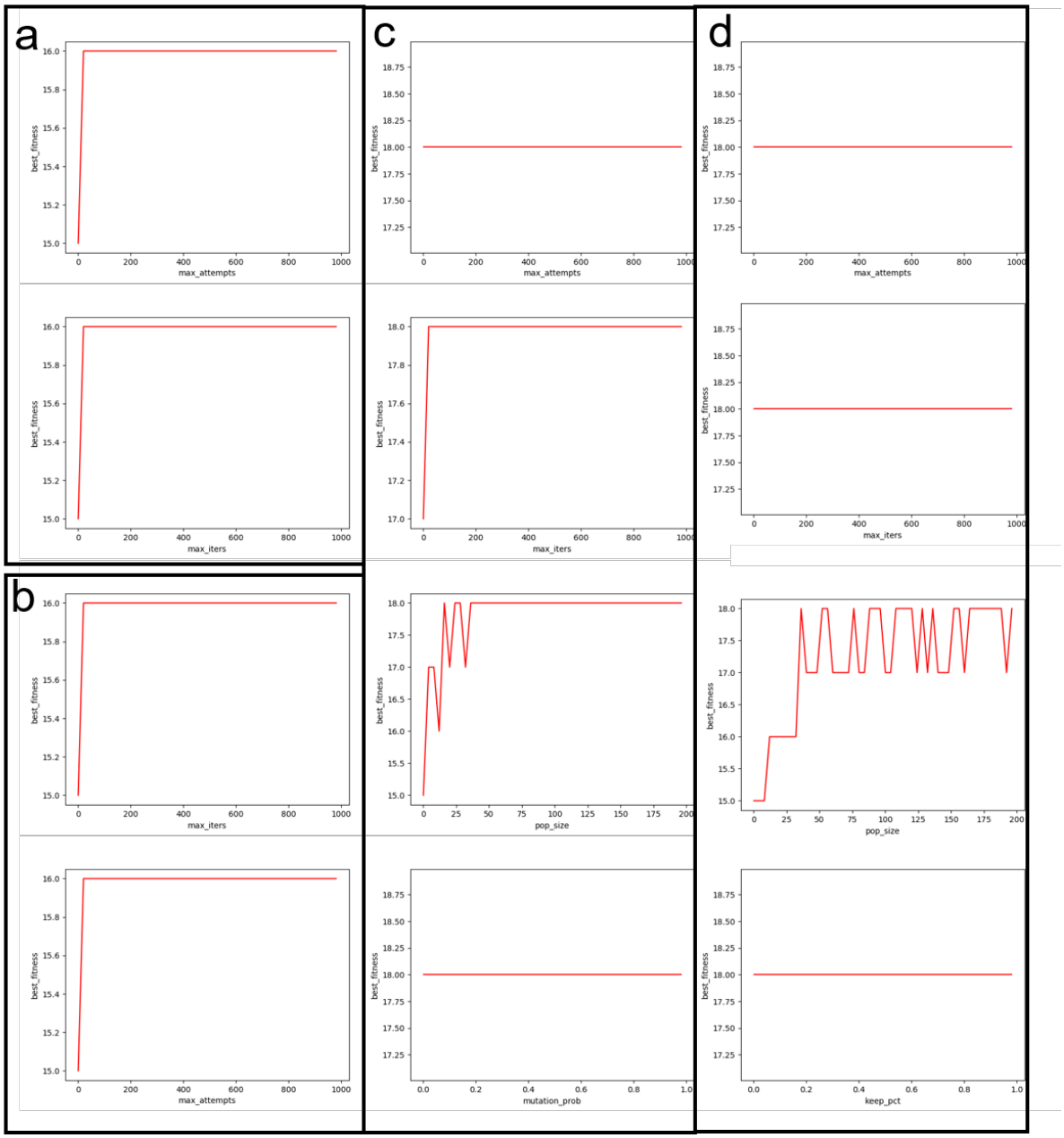

Figure 1: Parameter optimization for the continuous peaks problem. a) `max_attempt` and `max_iteration` for RHC. b) `max_attempt` and `max_iteration` for SA. c) `max_attempt`, `max_iteration`, `pop_size` and `mutation_prob` for GA. d) `max_attempt`, `max_iteration`, `pop_size` and `keep_pct` for MIMIC.

3.2. Algorithm comparison

Using the optimized parameters, four algorithms are compared by running time and fitness score. As shown in **Figure 2**, SA is highlighted when solving TSP, as it significantly (i.e. inter-algorithm difference larger than standard deviation) outperforms all other algorithm on both (highest) fitness score and (lowest) running time, on all sample sizes. It is noticeable that RHC performs worst in terms of fitness score, which is reasonable, as RHC lack the ability of exploration. For many scenarios, the best fitness score RHC can give out is actually derived from local optima, rather than the global optima. On the other hand, 8-city TRP is not very complex, i.e. the local optima is not very shallow, as the map of 8 cities is pretty much

fixed. Thus, compared with GA and MIMIC, which iterates slowly, SA will have an advantage of being able to iterate many times quickly (in term of clock time) and find the global optima.

Table 2. Optimal parameters adopted for algorithm comparison study.

Algorithm	**Parameter**				
	max_attempt	*max_iteration*	*pop_size*	*mutation_prob*	*keep_pct*
TSP					
RHC	*20*	*50*			
SA	*1*	*600*			
GA	*10*	*10*	*100*	*0.1*	
MIMIC	*10*	*10*	*100*		*0.1*
Cpeaks					
RHC	*10*	*10*			
SA	*10*	*10*			
GA	*10*	*10*	*100*	*0.05*	
MIMIC	*10*	*10*	*100*		*0.05*
KP					
RHC	*10*	*10*			
SA	*10*	*10*			
GA	*10*	*10*	*100*	*0.1*	
MIMIC	*10*	*10*	*100*		*0.1*

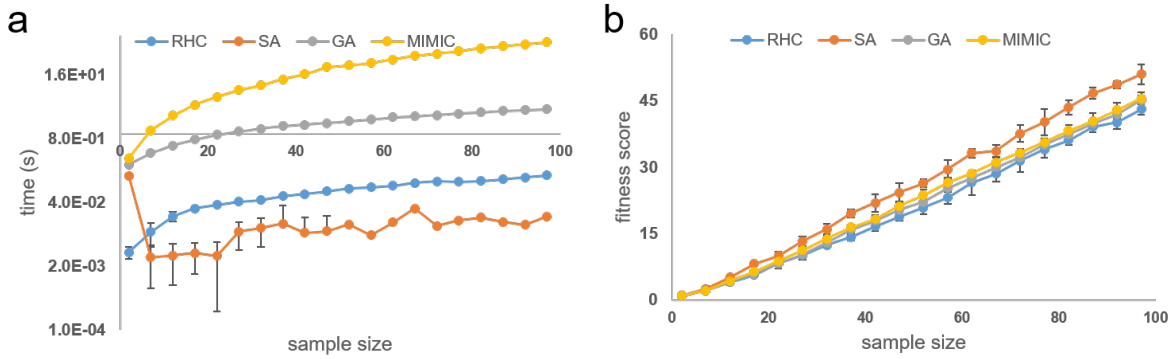

Figure 2. Comparison of algorithms on TSP of various sample size. a) Algorithm performance on time. b) Algorithm performance on fitness score. n=10.

As shown in **Figure 3**, GA is highlighted when solving Cpeaks, as it significantly (i.e. inter-algorithm difference larger than standard deviation) outperforms all other algorithm with highest fitness score on most of the sample sizes. Compared with TSP, which has a given map of all cities, Cpeaks is a more "tricky" problem. Cpeaks requires the optimization algorithm to achieve higher score by creating the longest contiguous bits of 0 or 1, in any part of a bit array of given length. The lack of relationship between the actual value of a certain bit and the fitness score made the "exploitation"-focusing algorithms (i.e. RHC and SA) perform extremely poor. As shown in **Figure 3b**, as expected, although each iteration take very short amount of time, SA and RHC failed completely to find any way to improve

their fitness score. The fact that any bit can be either 0 or 1 to achieve highest score generated enough noise to confuse RHC and SA. On the other hand, the “exploitation and exploration”-focusing algorithms (i.e. GA and MIMIC) obtained much better fitness scores.

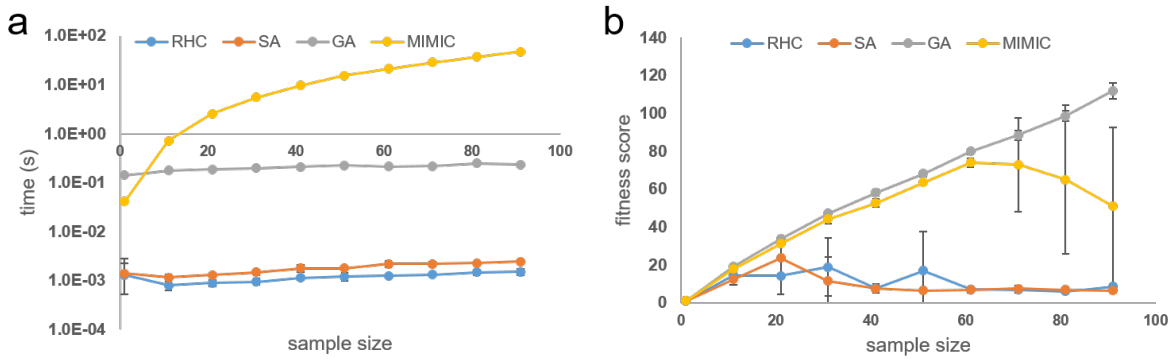

Figure 3. Comparison of algorithms on Cpeaks of various sample size. a) Algorithm performance on time. b) Algorithm performance on fitness score. n=10.

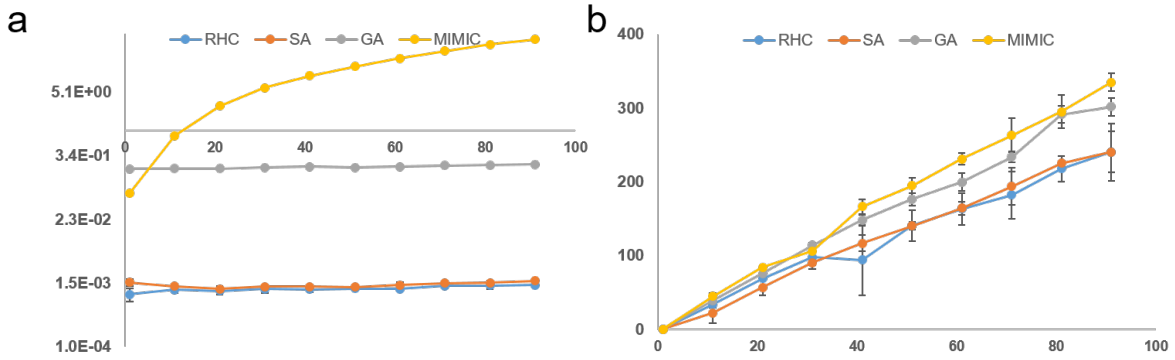

Figure 4. Comparison of algorithms on KP of various sample size. a) Algorithm performance on time. b) Algorithm performance on fitness score. n=10.

It is however worth to note that while GA and MIMIC perform similarly successful when the sample size is smaller than 60, GA outperformed MIMIC when using larger sample size. I reason that the downfall of MIMIC on larger sample size is actually due to MIMIC’s ability of adopting the structure of investigated data over generations. For example, in Cpeaks problem, a 5- bit array [0, 0, 0, 0, 0] will generate a much higher score compared with a 5- bit array [0, 0, 1, 0, 0], as the former contains a longer contiguous bits. In MIMIC algorithm, this observation will make the algorithm “remember" that using 0 on the 3rd bit will outperform than using 1. That is, the probably distribution on the 3rd bit of this array shifted toward of using “0”. However, remembering such information worth nothing, other than overfitting the data.

As a result, while GA outperforms MIMIC when optimizing for Cpeaks, in which remembering the structure of data always means overfitting and thus harmful for the optimization process. However, when optimizing for the Knapsack problem, remembering the structure of data (e.g. remember which item has more value per weight) is useful. Thus, MIMIC is highlighted when solving KP (**Fig. 4**), as it significantly (i.e. inter-algorithm difference larger than standard deviation) outperforms all other algorithm with highest fitness score on most of the sample sizes.

Other than the three problems that I have described above, I have also investigated other problems, such as OneMax, k-Max color, and flip-flop. In terms of fitness score, MIMIC usually performs well (if not best), while the running speed (per iteration) thereof cannot compete with RHC and SA. I realized that there is a trade-off between computational cost and time. MIMIC passes most information between iterations, thus is the most computational expensive algorithm, but performs best in terms of fitness. On the other hand, RHC and SA are computational in-expensive and runs fairly quickly, but in the cost of that they cannot handle complex optimization problems.

4. Conclusion

In this part of report, I have compared four different random optimization algorithms on three problems. The biggest lesson that I learned is, for all algorithms, these exists a trade-off between computational cost and running time. You have to sacrifice one side of the trade-off for the other side. After all, “there is no free lunch”.

Part 2. Algorithm optimization over different problems

1. Introduction

In this part of the report, I will investigate the performance of three algorithms, i.e. RHC, SA, and GA, on random optimization of on Artificial Neuron Network (ANN). Both problems and algorithms are adopted using mlrose [1]. As a gold-standard algorithm, a gradient descent (GD) algorithm is also compared. In this part of the report, I will firstly find optimal parameters for each algorithm, then compare the performance of them.

2. Experiment design

2.1. Dataset

The dataset Epileptic Seizure Recognition Data Set (ESR) [4], which has been analyzed in Assignment 1, is adopted in this assignment. This dataset is collected as the recording of brain activity via electric signals. Each row of the dataset describes 178 data points (i.e. X1-X178) happened within 1 second in a person's brain. Each row is associated with a label y that represent 5 different status (eyes open, eyes closed, healthy area, tumor area, and seizure) of the person tested. All of the five kinds of y status have equal proportion in this dataset. For this assignment, the first 6000 rows in the dataset are split into a 5000-row train and 1000-row test dataset. The total size of the training and testing dataset is 5.5 MB. The dataset is separated into the training dataset and the testing dataset. The former will be used for parameter optimization, and the latter will be used for algorithm comparation.

2.2. Parameter optimization

An ANN using one hidden layer of 50 nodes is adopted from **Assignment 1**. To find the best performance, parameter optimization is performed over all four algorithms (RHC, SA, GA, and GD) to find optimized parameters. Similar to what has been described in **Part 1, Section 2.1**, while keeping all the other parameters at default, one of the parameters are changed, and the resulted accuracy score, for both the training set and cross-validation set, is plotted against the changing parameter. For each parameter, 50 different values are tested on the training and the validation set to find the optimal parameter setting. The value of the parameter that grant the maximum cross-validation score is adopted as optimal. The cross-validation is performed using 10-section shuffle-split. Finally, using the optimal parameters, the performance of all four algorithms are compared on the testing set.

3. Results and Discussion

3.1. Parameter optimization

As shown in **Figure 5a**, for ANN using algorithm GP, the parameter that affects the performance most is `learning_rate`, which maximizes the performance for both training and the validation score at the value of 0.00032. Additionally, `max_attempts` and `max_iter` need to be larger than 20 and 200 for best performance, respectively (**Fig. 5b-c**). These parameter values are then taken as optimal. Under such parameters, GP reaches highest accuracy of 60.8% on the validation set, which is similar to what we have seen on ANN using back-propagation. For other algorithms, similar process on parameter optimization has been made. The optimized hyperparameters are listed in **Table 3**. Note that although increased `max_iter` could result in higher training accuracy, I adopted the `max_iter` value when the validation curve plateaus. Other than GP, however, the randomized optimization algorithms all suffer from high bias /low variance problem, as the accuracy of the training set cannot even overpass 0.3 (data not shown). This indicate that

the optimizing for ANN is a much more complex problem compared with sample problems such as TSP in part 1. To move the training and validation curves “upwards”, more powerful and complex random optimization algorithms is needed.

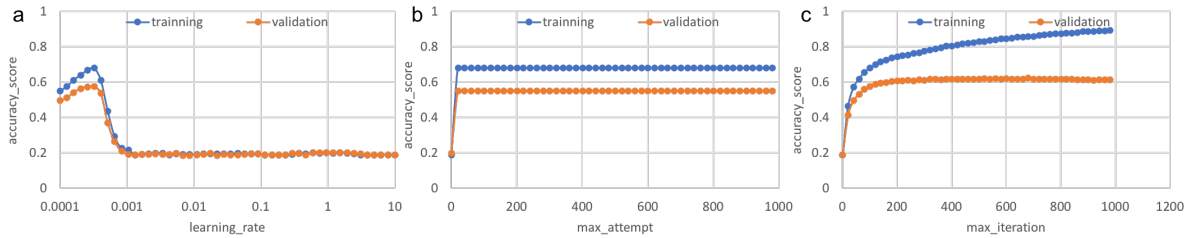

Figure 5. Parameter optimization of GP on ANN. a) accuracy score on `learning_rate`. b) accuracy score on `max_attempts`. c) accuracy score on `max_iters`.

Table 3. Optimal parameters adopted for algorithm comparison study.

Algorithm	**Parameter**				
	max_ attempt	*max_ iteration*	learning_ rate	*pop_ size*	*mutation_ prob*
RHC	*1000*	*500*	*10*		
SA	*1000*	*500*	*10*		
GA	*50*	*200*	*10*	*200*	*0.26*
GP	*20*	*200*	*0.00032*		

3.2. Inter-algorithm comparison

Using optimal parameters, the algorithms tested. The resulted performance score and testing time is shown in **Table 4**. The running time of GP, RHC, and SA are relatively short, while GA is rather computational expensive (~2s vs. ~200s and above). This is reasonable, as compared with other algorithm, GA has a population size of 200. This effectively means that for each iteration, GA would require 100 + fitness evaluations, while other algorithms only need 1-3 evaluations.

Table 4. Inter-algorithm comparison on performance accuracy score and running time, as average of 10 trials.

Algorithm	**test_accuracy**	**time (s)**
GD	0.79	2.04
RHC	0.25	2.02
SA	0.25	2.92
GA	0.27	194.97

GP significantly overperforms other algorithms on accuracy score. Considering that this dataset has five values on y, and each of them has a chance of 0.2 to appear, we can say that RHC, SA, and GA barely outperforms random choosing. My reasoning to this observation is that, when RHC, SA, and GA are looking for global optima (given that they are not stuck on local optima, which is quite likely), they mostly reach the global optima in a randomized manner. That is, even when they reach the global optima, without knowing their surrounding fitness gradient, the knowledge solely on global optima will not be that useful.

However, if there are enough iterations/data size given, I would expect that at least GA should be able to find the path to higher accuracy score on this ANN.

4. Conclusion

In this part of the report, I did not find any randomized optimization algorithms settings that could optimize neural network parameters in a meaningful manner. Compared with the gold standard, i.e. gradient descent baseline, the ANN is still too complex. All randomized optimization algorithms are suffering from a very high bias (featured with low prediction accuracy on training set) problem.

References (for both Parts)

[1]. Hayes, G., 2019. mlrose: Machine Learning, Randomized Optimization and Search. Retrieved from https://mlrose.readthedocs.io/en/stable/
[2]. Baluja, S. and Caruana, R., 1995. Removing the genetics from the standard genetic algorithm. In Machine Learning Proceedings 1995 (pp. 38-46). Morgan Kaufmann.
[3]. Baluja, S. and Davies, S., 1997. Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space (No. CMU-CS-97-107).
[4]. Epileptic Seizure Recognition Data Set (2017, May 24). Retrieved from https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition