Assignment 2: Randomized Optimization

[Mingrui Sun]
[msun333@uga.edu]

I. Introduction

A big chunk of machine learning in real world is about function approximation where ideal representation of data being retrieved through optimization of different model parameter using the training data. Depending on the complexity of the model used(how many weights and bias in the system), this optimization problem could be quite a challenge. For example everyone's second best model-Neutral Network(NN) would require more parameter optimization than Linear Regression model. As a practitioner of ML in the field, forming better understanding of these randomized optimization methods would goes a long way.

In this work, Randomized Hill Climbing(RHC), Simulated Annealing(SA), Genetic Algorithm(GA), MIMIC ([2]) were implemented using python mlrose package.([3]) The benefit of more advance algos(GA,SA,MIMIC) were highlighted using three distinct maximization problem in section I.

In section II, three NN(Neural Network) model was built in the exact same configure as in HW 1 with the same datasets(UCI Adult Earning Dataset)([5]), but instead of using gradient descent when training the model, we apply RHC,GA,SA to optimized all weight and bias in the network. Performance statistic of these NN model built were discussed.

II. Section I

In this section, all result of different optimization algo in different maximization problem would all a average of a 100 runs with random seed 0 to 99. This ensure that the result and ramification found here have good generalization property across the space. In each problem, the max attempts(restart number) and max iteration setting were set to the same for each algo to ensure a apple to apple comparison. Also early stop of each algo was all enable. For Simulated Annealing Algo in this section, if not stated otherwise, a default exponential decay schedule was used for temperature profile.

A. *N Queens Problem-Simulated Annealing*

The idea behind N queens problem is fairly straight forward which a bit of chess background. In chess, queens can move/attack in both diagonal, horizontal and vertical direction. N queens problem is how can N queens be fitted in a NxN chess board without any pairs of queens can attack each other. For this report, we used a custom fitness function that return the number of non-attacking queens pairs for each board configuration. For example for 8 queens problem, the perfect solution would have a fitness of 7 + 6 + ... 1 = 28 and for 16 queens it would be 120. This custom fitness function change N Queens problem from a minimization problem to a maximization problem.

For this report, N Queens problem was used to highlight the power of SA algo. For this problem, evaluation of the fitness function is cheap. Also the problem have multiple solution that all have the same fitness value so the all local maximum in this case is global maximum. i.e. 12 out of 92 solution($\sim$ 13%) in 8 queens problem is perfect and 1846955 out of 14772512 solution($\sim$12.5%) for 16 queens is perfect.([1]) Combine this two features, a optimization algo that explores large region fast and climb a bit to refine solution would reach a perfect solution faster as we don't need to worry about local maximum. SA algo fits exactly to this requirement: at high temperature it explore large region of parameter but refine the solution at low temperature.

Result of these four algos on 8 queens and 16 queens were shown in figure 1,2,3,4 and table 1. Figure 1,3 shown the fitness curve of all four algos. Figure 2,4 shown the run time statistic of different algos in each problem. For the fitness curve, both problem shown similar trend as the more advance population based algos GA and MIMIC coverage faster in a far less iteration than simpler single pointer algos SA and RHC. In GA and MIMIC, in each iteration, more data points are being evaluate so in the ends these algo would converge faster to higher fitness than SA or RHC. Also in table 1 which shown the percentage of tries that generate perfect solution, GA and MIMIC shown far higher success rate than RHC and SA. Wells so base on these result GA and MIMIC are superior right? When looking at figure 2,4, things start to get interesting as RHC and SA have runs time two magnitude faster than GA and MIMIC. This means that in a time limit use case(most real world application), we can dramatically increase the max iteration and restart number be make values in table 1 looks better when the run time still in a similar range to GA and MIMIC!

So run time result of SA and RHC with 100 times more maximum iteration and restart setting(both from 100 to 10000) were shown in figure 5 for 16 Queen problem. Both SA and RHC run time cluster below 1min when RHC produce consistent shorter run time than SA(also shorter than GA and MIMIC run time in figure 4). This run time difference was mostly due to extra steps in SA like calculating probability of explore needs to be done in each step. The percentage success for SA and RHC in this case were 96% and 3% respectively. Though RHC have shorter run time, SA ends up producing consistence success in 16 Queens problem. SA and RHC are both single pointer algorithm. By single pointer i means that in each iteration, only a single data point is evaluate unlike population based algo(GA,MIMIC) that a population

of points were evaluate. The added simulate annealing steps in SA enable more efficient exploration through out space which eliminate the dependent of luck(selection of starting point/random seed) in even a simple question like N queen.
Overall, result here shown the power of SA in a simple problem which a bunch of perfect solution like N-queens. Though SA have extra knob to turn like the setting of temperature decay profile which we can fine tune the ratio of exploration to climbing in each iteration, for N-Queens problem, we achieved great result without even touching it. Fast run time and not much tuning require would be the major benefit when comparing SA to GA and MIMIC. Thanks to fast runtime, majority of the time SA can just brute force it way out of any problem(by brute force i mean increase number of iteration and restart number).

Table 1.Percent of Try Reach Perfect Soulution		
Algos	8 Queens	16 Queens
RHC	8	0
SA	1	0
GA	91	12
MIMIC	33	0

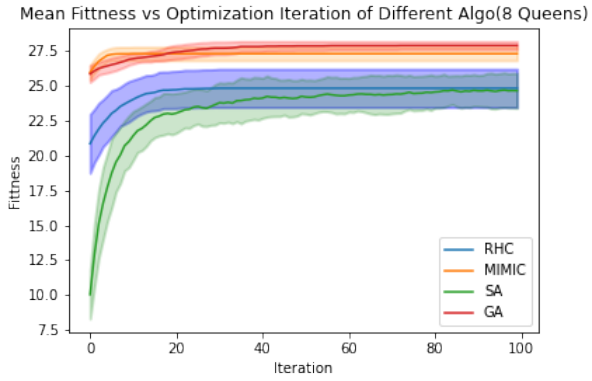

Fig. 1. Fitness Curve on 8 Queens Problem

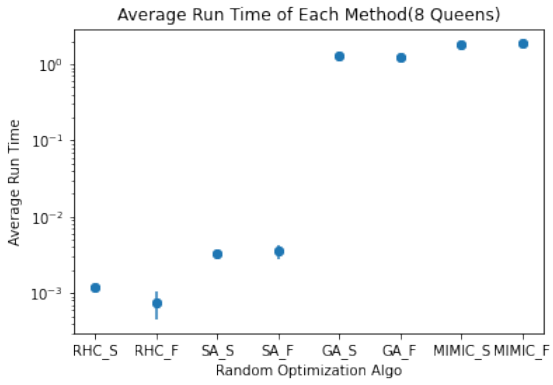

Fig. 2. Wall Time Statistic in 8 Queen Problem, S and F here stand for the success/failed to achieve the best solution.

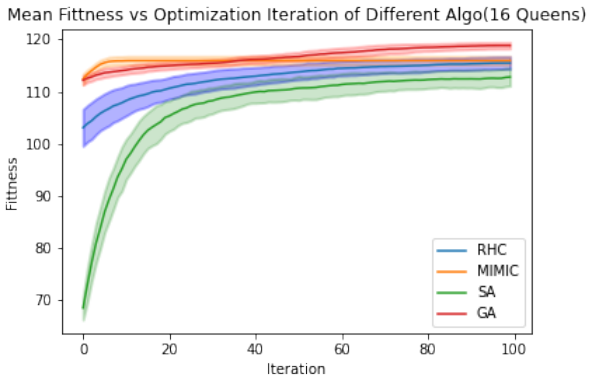

Fig. 3. Fitness Curve on 16 Queens Problem

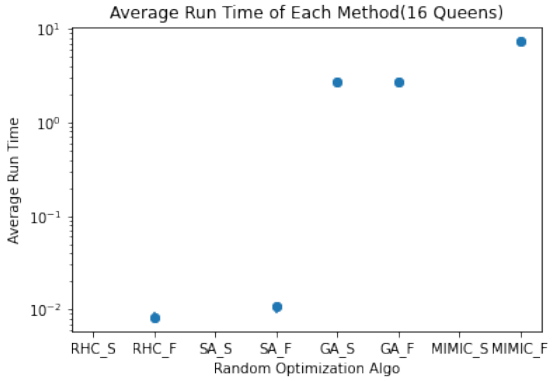

Fig. 4. Wall Time Statistic in 16 Queen Problem, S and F here stand for the success/failed to achieve the best solution.

B. Traveling Salesman Problem(TSP)-Genetic Algorithm

Traveling Salesman Problem is a NP hard optimization problem. It can be describe as the following: Given a list of cities/towns and their distance to each other, how to output a route that go through all cities/town with the lowest amount of total distance travel.[4] In this section, since we focus on maximization problem for this report, the original fitness function of traveling salesman problem was changed to the negative of the total distance travel.
Unlike the in N queens question that i can point out the exact number of fitness value and number of solution exist(NP complete), for TSP it the solution number is not so clear and it is hard to know exactly the best solution without actually doing the optimization. This difference make the more brute force like method like SA and RHC really unlikely to achieve the best solution compare to GA or MIMIC in this problem as local maximum do exist in the space. In TSP optimization, it is more important to have a lower total distance(high negative distance) than having lower optimization run time like in last section.
In this work, only a 8 cities version of TSP was generated to test performance of different algo. Result of these four algo were shown in figure 6, 7. In figure 6, GA and MIMIC

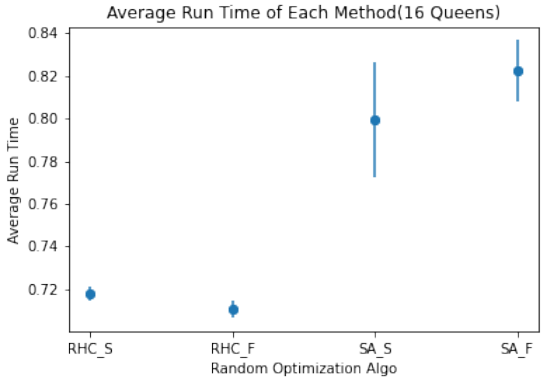

Fig. 5. 100 Times more Max Iteration and Restart Number's Run Time for SA and RHC

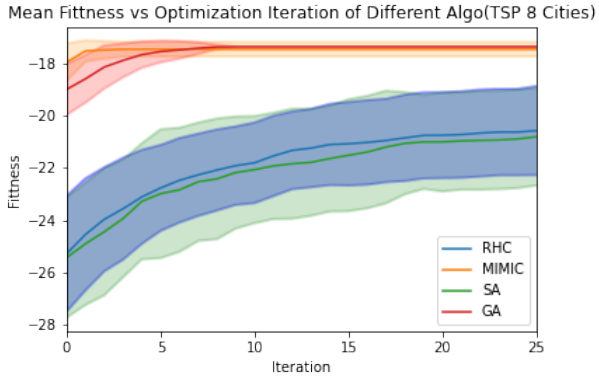

Fig. 6. Fitness Curve of Four Optimization Algo(TSP 8 Cities)

converage to a high mean fitness than RHC and SA. MIMIC seems to reach the a high fitness earlier than GA but it ends up having larger spread of fitness value and lower fitness mean in the end. Judging only by the end result at 100 iteration(not shown in the graph), the GA produced higher mean fitness with lowest spread(std of the mean). In fact, the spread of GA's result final fittness have spread way less than spread of the MIMIC result(1e-14 vs 0.28). This means the GA in this problem not only produce the best solution, but did it in such a consistency that the second best algorithm(MIMIC) cannot keep up. Figure 7 shown the runs time difference of this algo. Just like in N Queens(figure 2), RHC and SA out performed population based method GA and MIMIC in terms of run time in about two magnitude. Also GA have constantly lower run time compare to MIMIC. Overall, these result shown that GA have superior performance in 8 cities TSP than the rest of algorithm in the work.

In additional to the stock performance shown before, hyper-parameters' impact on GA performance on 8 Cities TSP were also examined here. I focus on one particular parameter, the population size. The impact of population were shown in figure 8 and 9. As population increase, the fitness curve of GA converge faster as higher population means that we are more likely to get lucky(get close to the optimal solution) right from the start. But this fast convergence came in a cost of longer run time as shown in figure 9. From 100 to 300, the means of run time and spread of the mean almost increase linearly. In short for population size, a larger population size is always more favorable for GA as long as your machine can run it within your run time requirement.

C. *Max K Color Problem-MIMIC*

In this section, Max K Color problem was used to showcase power of the MIMIC algorithm. Max K Problem is a NP completed problem that ask for the maximum number different color can be assign to a map where no adjacent block have the same color.([6])

So far we already highlight the benefit of SA and GA using

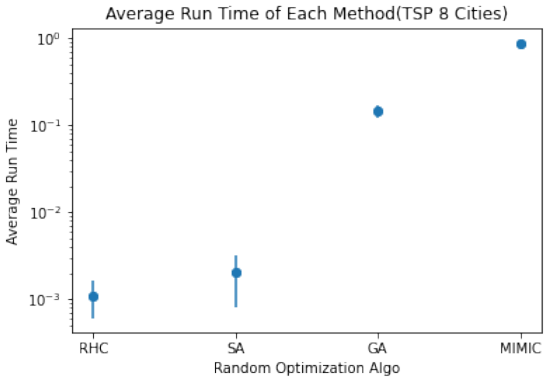

Fig. 7. Run Time Comparison(TSP 8 Cities)

N Queens Problem and TSP respective. Looking back to the fitness curve shown in figure 1 and figure 6, one common features between them is that though MIMIC wasn't the algo that ran fastest or produce the best result, it was the first algorithm to converge with the less amount of iteration. So in what world would MIMIC perform the best? Well when the evaluation of fitness function is so complex that it is really expansive to calculate fitness then MIMIC would produce superior performance. I cant really came up with a optimization problem with fitness function that complex, but i can simulated the same experience by limiting the max iteration value of all algorithm to a small number 3(compare to 100). Imaging this being in a world that we are not being charge by the server use time but the actual number calculation the program runs.

A 8 edge Max K Color problem with K = 6 were used in this section. For all optimization method used here, the restart number and max iteration number were limit to 5 and 3 respectively. The restart number was limited as i would like all algorithms here to have a low chance of getting lucky(start close to the maximum). Result were shown in figure 10, 11. As fitness curve figure 10 shown, MIMIC ends up having higher mean fitness and lower spread compare to all other methods. GA came close but overall it achieved lower mean fitness and

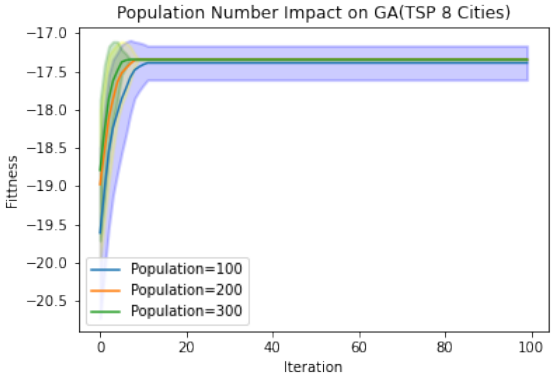

Fig. 8. Population Size Impact on GA Fitness Curve(TSP 8 Cities)

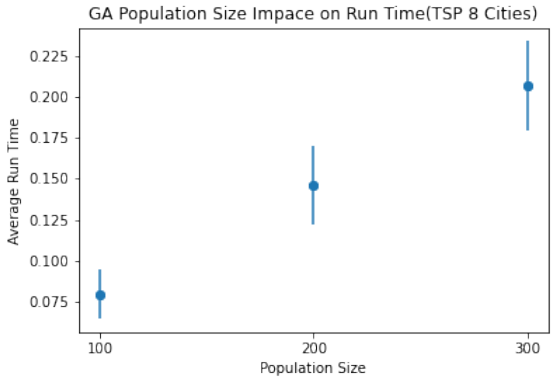

Fig. 9. Population Size Impact on GA Run Time(TSP 8 Cities)

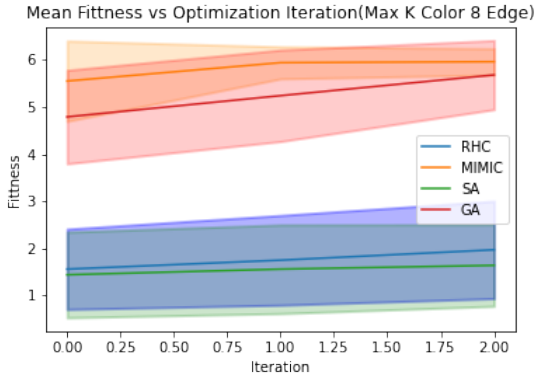

Fig. 10. Fitness Curve in Extreme Limit Iteration Setting(Max K Color)

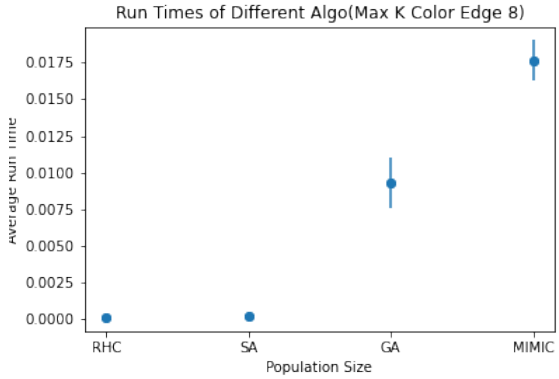

Fig. 11. Run Time of Different Algos(Max K Color)

have larger spread on the mean in the end. Runs time statistic were shown in figure 11, though MIMIC have the best fitness wise result, its run time is the worst among all algorithms testing here. Overall, only in case that fitness function evaluation process is too expansive then MIMIC would then have a place compare to especially genetic algorithm. Result shown here are consistence with Dr.Isbell original MIMIC paper analysis that shown MIMIC consistently converge faster(iteration wise) compare to other algorithm. The reason why MIMIC converge in shorter iteration compare to especially GA come down to the process of recalculate the likely hood distribution and re-sample population based on that. This is a more efficient way of utilizing the current population(i means estimation of points) fitness information than crossover in GA where that mostly happened by lucky(crossover). This make each iteration in MIMIC have higher information gain than in GA which ends up causing it to converge faster. But all benefit come in the caveat that the likely hood density(distribution) estimation could really be approximated right and also this MIMIC may take forever to run. These caveat kind of make MIMIC a great choice for some special problem but failed to reach the popularity of GA and SA in the field.

III. Section II-Neutral Network Meet RHC,SA,GA

With the strong points of each algorithms figure out, now we can actually apply these method to a real world problem-optimize weight and bias in a Neutral Network. For this work, the dataset i choosed was the UCI Adult Income Dataset as in HW1. In HW1, it was already shown that the best MLP hidden layer structure for this problem would be a [16] so the same network structure would be reused. This choice of a single hidden layer structure also release some work load from the weight/bias optimization problem as more complex NN structure would result in a larger parameter space(harder optimization problem). From post experience, sklearn and mlrose implemented NN model was trained only utilizing CPU which have limited parallelization capability due to limit number of core/thread it have compare to a GPU where most modern NN network train on. This means every training in this section was heavily CPU bounded and can take hours to runs. So unlikely in section i we have the privilege of really observing each optimizes' behavior with 100 different random seed, i choose to only used one random seed 3 and relay on mostly restart(max attempts) in this section. In this section, four distinct NN models were built from the same training set but with RHC, SA, GA optimization and gradient descent(the optimal way). Since the optimizer used was chanced from

HW1(we used Adam) to here, fine tuning of learning rate was done for each optimizer NN to achieve the best performance. Performance of these final tuned NN were compare using the same test set to show difference between them.

A. *RHC's NN Learning Rate Tuning*

Learning curve of NN optimize with RHC optimizer with different learning rate was shown in figure 12. As learning rate increase, the loss drop faster so lower loss was achieved in less iteration. 0.5 learning rate achieved the best log loss so this learning rate was used for the final RHC NN model. Noted that normally for NN training using adam or Stochastic gradient descent(SDG) optimizer, a learning rate higher than 0.1 would be unusual as those method are more efficient to drop down loss so lower learning rate is needed to ensure them not overshooting. But in our RHC case, it is really prone to problem of local minimization, so a high learning rate can be appropriate as it is more likely in higher learning rate to escape those case.

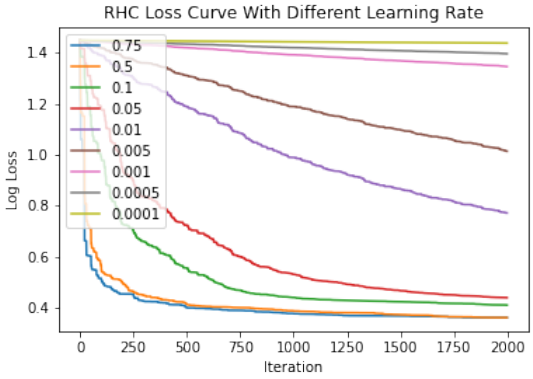

Fig. 12. Learning Rate Impact on the RHC Loss Curve

B. *SA's NN Learning Rate Tuning*

Base on what we learned in section I, a great benefit of SA optimizer is its low runs time compare to population based method like GA and MIMIC. High speed means that for SA, we can use really aggressive setting(high max attempts and max iteration) without much run time increase.
Learning curve of NN optimize with SA optimizer with different learning rate was shown in figure 13. Just like in RHC section, a larger learning rate 0.5 produce the lowest loss in the end. Comparing figure 13 to 12, we can clearly see that at low iteration the SA loss curve have more noise(not smooth) than RHC loss curve at the same learning. This coming from the fact that at low iteration number, SA optimization was in the low temperature phase which cause more random exploration through out space which lead to noisy loss. Overall, a 0.5 learning rate was selected for our final SA's NN model.

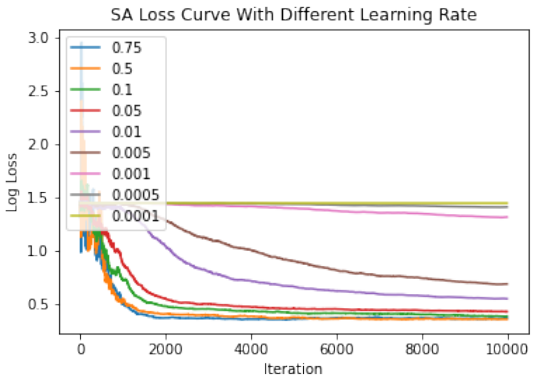

Fig. 13. Learning Rate Impact on the SA Loss Curve

C. *GA's NN Optimizer Parameter Tuning*

Since learning rate have no impact on GA as it is not a step wised base method, so in this section the power of population size was discussed. Population sizes impact on learning curve of NN optimize with SA optimizer was shown in figure 14. As population size increase, the learning curve reach lower loss in the end. High population leads to better result as it give a better representation of the overall loss function space. At higher population setting, majority of the space is already explored so GA dont needs to relay on luck control mutation to explore. This make GA less likely to stop at unfavorable local minimal. Another interesting finding is that all GA learning curve have a staircase shape compare RHC and SA's mostly smooth curve. This staircase behavior was due to the GA not being a step wise method. It also indicate that GA can high volatility to converge: it is hard to know whether we converge or not at each staircase level(it is mostly flat). Overall for the final model, we want to use the biggest population size that our run time budget can afford, so in the end a 1000 population size was selected.
In section 1 I praised GA a lot for its consistence performance for those simpler optimization problem, in real world application like NN weight/bias optimization with limited time, it is impossible to fine tune all GA's hyper parameter(population size, mutation rate, percentage breeding, iteration number) as running GA optimizer itself is a time consuming process. GA is only a great choice if getting the optimal solution is the only concern.

D. *gradient descent's NN Learning Rate Tuning*

Finally we come to the big gun, the actual better optimizer for NN, the gradient descent. In real world application a version of gradient descent - stochastic gradient descent was more commonly used as optimizer for training neutral network. Just as the name implyed, in GD, optimization was done by repeatedly going down the opposite of the approximated gradient of loss. Since we always changed weight/bias to the steepest gradient, a faster convergence to a minimal compare

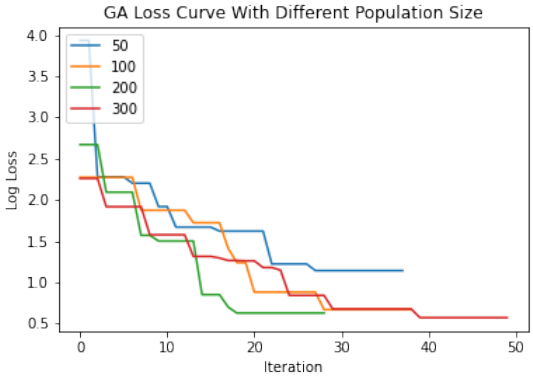

Fig. 14. Population Size Impact on the GA Loss Curve

to in RHC were only loss value was compare. Due to this feature, the fine tuning of learning rate for GD become more important.

Learning curve of NN optimize with SA optimizer with different learning rate was shown in figure 15. Unlike for RHC and SA that a large learning rate lead to lower end point loss, smaller learning rate like 0.0001 or 0.00001 lead to better loss. When learning rate too high, the GD optimizer stuck at a local minimal which cause all those learning curve to be a flat straight line. In the end the 0.00001 learning rate was selected for the final GD's NN model as it reached the lowest loss in the end and the learning curve being more well behaved(continuous smooth curve).

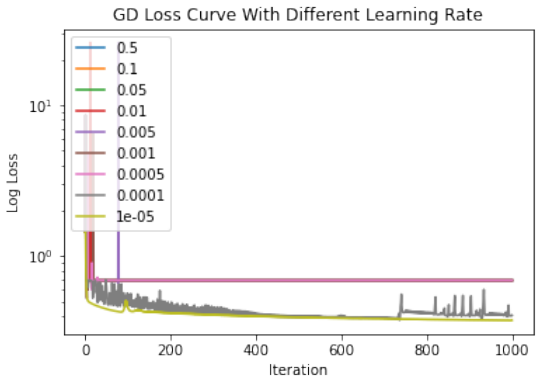

Fig. 15. Learning Rate Impact on the GD Loss Curve

E. Final Comparison of Tuned NN model Trained

Performance of these resulting NN model using different optimizer were shown in figure 16, 17, 18. Since RHC and SA are faster method, more aggressive restart value(10000) were used for them(compare to 10 for GA and GD). As shown in figure 16, GD and SA converge and stop at lower iteration(sub 10 iteration) compare to SA and RHC. The ending loss of these optimizer can be rank in order of RHC $<$ SA $<$ GA $<$ GD. This is consist as the benefit of GD and GA stated before. Training time of these optimizer was shown in figure 17. Not surprisingly, GD outperform all other optimizer in about two magnitude when the rest rank in orders of GA $>$ SA $>$ RHC. Though result here are only from one seed, restart option was enable for all optimizer so the result shown in this section still can be used to generalized performance of this method. Finally the prediction performance of all NN model trained were shown in figure 18. Surprisingly, in terms of both in/out sample prediction score, simpler method like RHC and SA actually outperform GD and GA. This may be due to the choices made of using overly aggressive restart number used for both optimizer. Also for GD and GA, we don't really fined tune every parameter to improve their performance. Like what in section I shown, though SA and RHC maybe stuck at local minimal, if a huge number of restart value was allowed then that could be overcome by brute force when also maintain a reasonable run time. On the other hand, GA is a great optimizer but needs a lot of tuning and machine resource to run. So in real world use case, we should always tried the brute force like method SA and RHC first and only used more advanced(smart) method like GA and MIMIC as a last resort. Another interesting point is GD have superior training time and not bad predicting score compare to all other optimizer. For this work, the NN model is only have a single layer of hidden layer. When number of netrons and layer increase, RHC, SA, GA training time would all go astronomical fast then we would be always using a gradient descent based type of optimizer. This is why in real world we never use RHC,SA,GA to train NN model.

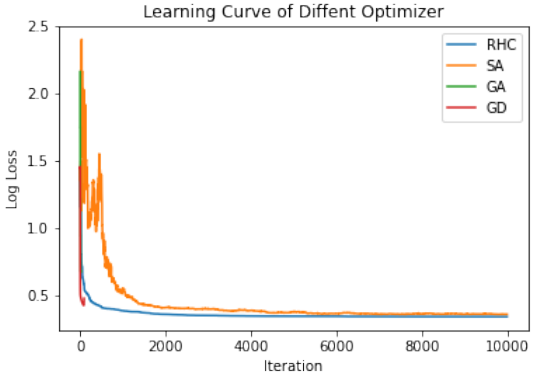

Fig. 16. Learning Curve of Different Optimizer

IV. References

References

[1] Durango Bill. *The N-Queens Problem*. URL: http://www.durangobill.com/N_Queens.html.

[2] Jeremy De Bonet, Charles Isbell, and Paul Viola. *MIMIC: Finding Optima by Estimating Probability Densities*. Ed. by M.C. Mozer, M. Jordan, and T. Petsche. 1996. URL: https://proceedings.neurips.cc/paper/1996/file/4c22bd444899d3b6047a10b20a2f26db-Paper.pdf.

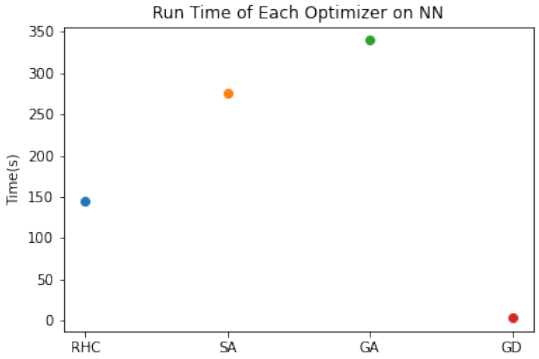

Fig. 17. Run time of each Optimizer

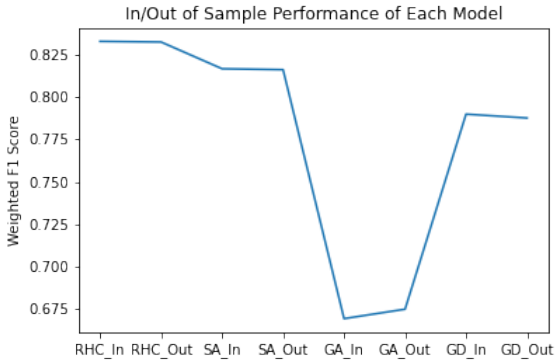

Fig. 18. Prediction Score of Final Model

[3] Genevieve Hayes. *mlrose: Machine Learning, Randomized Optimization and SEarch*. URL: https://mlrose.readthedocs.io/en/stable/.

[4] Genevieve Hayes. *TSP Problem-mlrose*. URL: https://mlrose.readthedocs.io/en/stable/source/tutorial2.html.

[5] R Kohavi. "Scaling up the accuracy of Naive-Bayes classifiers: A decision-tree hybrid". In: (Dec. 1996). URL: https://www.osti.gov/biblio/421279.

[6] Wikipedia. *Graph coloring*. URL: https://en.wikipedia.org/wiki/Graph_coloring.