Ashwin Rathie
CS 4641 – Machine Learning
Professor Byron Boots
6 November, 2018

Analysis of Randomized Optimization

In this assignment, we will explore and compare the capabilities of random optimization methods, namely the following algorithms: Random Hill Climbing, Simulated Annealing, and Genetic Algorithm. We will do so by first using them to tune the weights of a Neural Network being used on an existing dataset. Then, we will use them on two other optimization problems to draw conclusions about the "preferences" of these algorithms.

The Algorithms:

Randomized Hill Climbing (RHC):

Conceptually and in practice, RHC can be considered the simplest of the three optimization methods we are exploring. It works by looking at an arbitrary solution and incrementally moving towards a better solution by calculating the derivative at that point. It will continue to increment until it reaches a point at which no incremental change in its parameters would improve the fitness of the solution. At this point, we have hopefully reached the optimal solution. However, it is possible that the algorithms have gotten stuck in a local optimum and will not be able to find the global optimum. In order to combat getting stuck in a local optimum, random restarts are used, meaning that the algorithm will start over with a new random solution and move toward its local optimum. If this optimum is more fit than the current most fit solution, then we have found a better optimum, and hopefully, the global optimum. The number of restarts is the hyperparameter for RHC.

Simulated Annealing (SA):

SA works similarly to RHC, but with a core difference in its "decision making". While RHC can be described as "greedy", where it will always increment based on the best immediate improvement in fitness, SA will sometimes move to a less fit point. In accordance with the

analogy of metal “annealing”, the two hyperparameters of SA are cooling rate (CR) and temperature. Cooling rate is a constant between 0 and 1 that dictates the rate at which the temperature is decreased. A lower cooling rate generally allows for better results, but if it is too low, the algorithm’s time to run may become unnecessarily long. The temperature dictates how likely it is that the algorithm decides to move to a less fit point. A lower temperature means that it is less likely that SA will move toward a less fit point, and will just move toward the local optimum instead. Because we decrease the temperature over time, we could say that the strategy of SA is to ensure the solution doesn’t get stuck in a local optimum in the beginning but eventually move toward the best available solution. This is an advantage over RHC, but doesn’t mean that SA is not still susceptible to outputting a local optimum solution.

Genetic Algorithm (GenAl):

The genetic algorithm is an analogy for sexual reproduction. This is because, like in sexual reproduction, the genetic algorithm uses mutation and crossover to ultimately create a more fit offspring (solution). Crossover is when parts of several parent solutions are combined to create a new child solution. Mutation, the random changes in genetic information in nature, is the hyperparameter that controls the likelihood that part of the solution is randomly altered. In theory, this helps mitigate the danger of getting stuck in a local optimum. Mate is the hyperparameter that described how many newly created individuals (potential solutions) will be added per generation to replace “parents”. Population size (POP) is the size of the solution set that we keep and use to create new solutions. So, the three hyperparameters for GenAl are Mutation, Mate, and POP. GenAl can be beneficial over SA and RHC because crossover means that this algorithm uses solutions that we know perform relatively well to make better solutions, rather than just moving randomly.

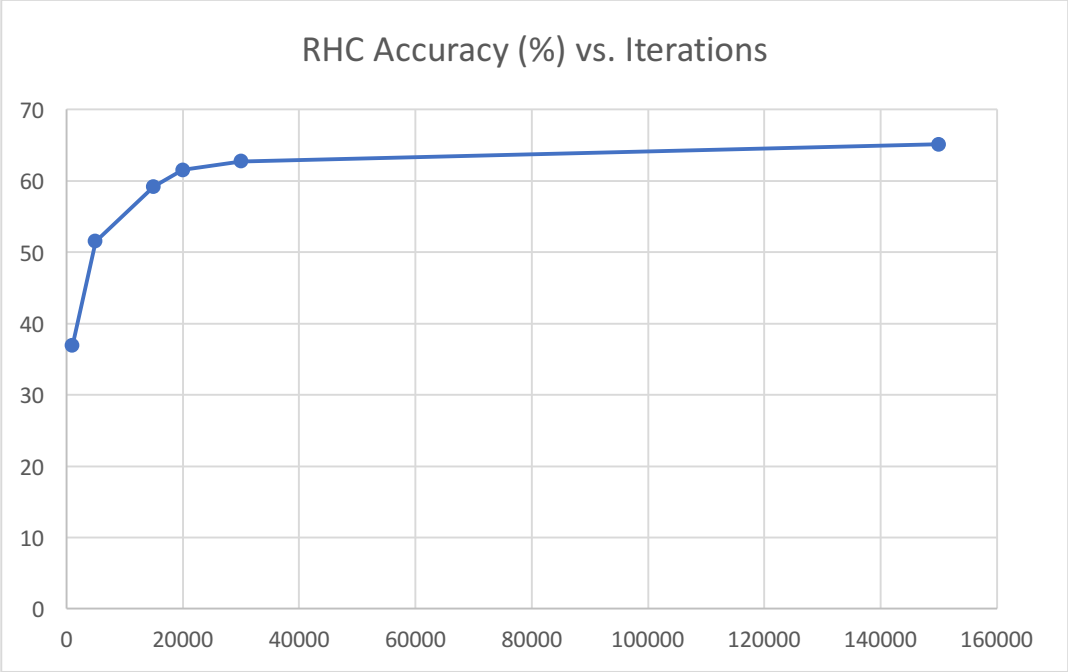
Part 1: Using Randomized Optimization to find Neural Network Weights

Part 1.0 – Introduction and Dataset:

The most widely used method for determining the optimal weights for a neural network is backpropagation, which is what was used in assignment 1. In this assignment, we instead use our optimization algorithms to tune the weights and observe the results. Based off of results from assignment 1, we will use a hidden layer size of 48. The Waveform-5000 dataset details 3 classes of waves according to 21 attributes with numerical values between 0 and 6.

Part 1.1 – Randomized Hill Climb:

ITERATIONS	% CORRECTLY CLASSIFIED	TRAINING TIME (SECONDS)
1000	36.94	73.238
5000	51.54	489.122
15000	59.21	870.767
20000	61.54	1475.529
30000	62.76	1974.77
150000	**65.13**	**11742.937**



RHC performs rather well. The best results that could be extracted from RHC was 65.13% testing examples correctly classified. However, this was achieved at a very high number

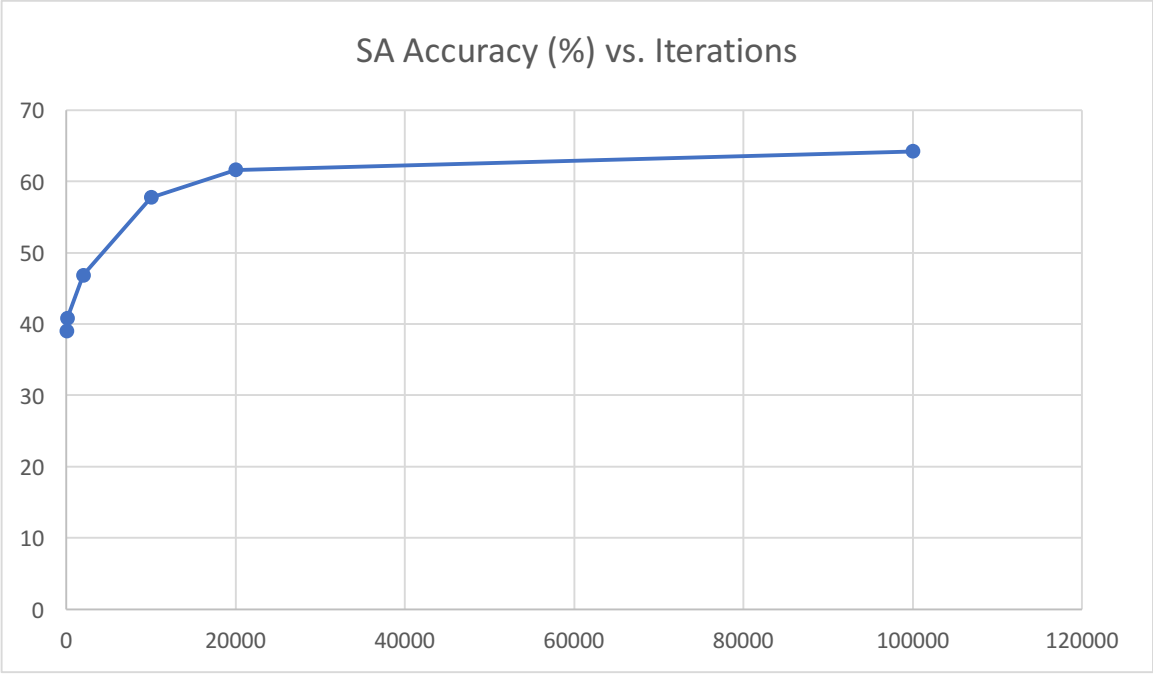
of iteration. As the number of iterations increases, we see that the accuracy improves but at a diminishing, logarithmic rate. Around 60% accuracy is reached in a relatively low amount of training time. RHC appears to get near its optimal solution quickly, which is a byproduct of the algorithm's conceptually simple and computationally non-intensive nature.

Part 1.2 – Simulated Annealing:

ITERATIONS	INITIAL TEMP	CR	% CORRECTLY CLASSIFIED	TRAINING TIME (SECONDS)
20000	50	0.9	60.78	1610.897
20000	1000	0.2	61.16	1275.91
20000	13	0.95	61.08	1438.426
20000	10000	0.9	60.62	1482.226
20000	1000	0.05	60.36	1399.293
20000	1000	0.8	61.99	1866.97
20000	1000	0.85	60.7	1866.97
20000	**1000**	**0.9**	**61.921**	**1400.014**
20000	1000	0.95	61.18	1339.774
20000	500	0.8	60.94	1580.789
20000	3000	0.8	60.6	1640.322
20000	3000	0.9	60.78	1335.288
20000	500	0.9	61	1544.913
20000	100000	0.9	60.96	1330.491
100000	1000	0.9	64.210403	6652.455

After deciding on an optimal initial temperature of 1000 and CR of .1, I ran it again keeping those hyperparameters constant.

ITERATIONS	INITIAL TEMPERATURE	CR	% CORRECTLY CLASSIFIED
10	1000	0.9	38.998
100	1000	0.9	40.769
2000	1000	0.9	46.845
10000	1000	0.9	57.733
20000	1000	0.9	61.622
100000	1000	0.9	64.210



SA's best score is around 64% at very high iterations, and around 60% in a reasonable amount of time. It appears to be able to achieve similar results as RHC in a comparable amount of time. SA's primary advantage over RHC is that it is better at not getting stuck in local optima and having a greater probability of finding the global maximum. However, if there are not many or no local optima that are not also the global max, this advantage diminished and they become very similar algorithms in practice. This appears to be the case for optimizing the weights of the neural network. This conclusion is further supported by the fact that changing the initial temperature and cooling rate hyperparameter appeared to have a minimal effect; if the hyperparameters that are designed to avoid getting stuck in local optima are not very important, then there could be a high likelihood of not getting stuck in a local optimum even if the hyperparameters were not in place.

Part 1.3 – Genetic Algorithm:

ITERATIONS	STARTING POPULATION	MATES PER ITERATION	MUTATIONS PER ITERATION	% CORRECTLY CLASSIFIED	TRAINING TIME (SECONDS)
3000	100	100	10	42.12	7348.225
3000	100	50	10	39.26	3683.999
3000	200	50	10	45.64	3841.561
3000	200	100	10	45.78	2589.217

3000	200	150	10	42.23	11725.801
3000	300	100	10	47.38	7359.904
3000	300	100	50	51.04	7928.938
3000	300	150	30	49.58	11809.306
3000	300	150	50	50.20	10950.91
3000	300	150	100	50.18	11494.917
3000	1000	100	50	51.92	7845.212
3000	**1000**	**100**	**150**	**53.76**	**13167.431**
10000	1000	100	150	62.93	37024.821

GenAl was able to achieve 63% accuracy, but at 37025 seconds! A more representative result is about 53%, which is notably worse than both SA and RHC. The accuracy increases with more iterations, but at a very diminished rate, so I've shown only the results of varied hyperparameters at 3000 iterations. The optimal hyperparameters appear to be POP = 1000, Mates = 100, and Mutations = 150. It should be noted that the training time for GenAl was, in general, much greater than the training times for both RHC and SA.

Part 1.4 – Comparison and Conclusion:

Out of the three algorithms, it is clear that GenAl is the worst when applied the problem of optimizing weights for our neural network. In GenAl, the potential weights make up the population and I believe it stands to reason that it shouldn't necessarily perform that well on this problem. The crossover portion of this algorithm takes portions of several solutions and combines them into a new solution. However, neural networks weights don't stand to benefit that much from this strategy. The improvement of the accuracy using GenAl is likely from the mutation aspect of the algorithm, which would have randomly changed the weights and tested to see if it yields better results, in a sense similar to a random restart in RHC. Intuitively, incrementing weights toward better fitness would work better than GenAl strategy.

As such, RHC and SA, which both increment a solution toward optima (or sometimes away from optima in the case of SA), performed significantly better than GenAl. Both resulting in scores around 60% in a similar time, it is difficult to determine which one is "better". In this case, it appears that there were not many local optima for the algorithms to become stuck in, devaluing SA's main advantage. However, because SA does have a greater ability to avoid

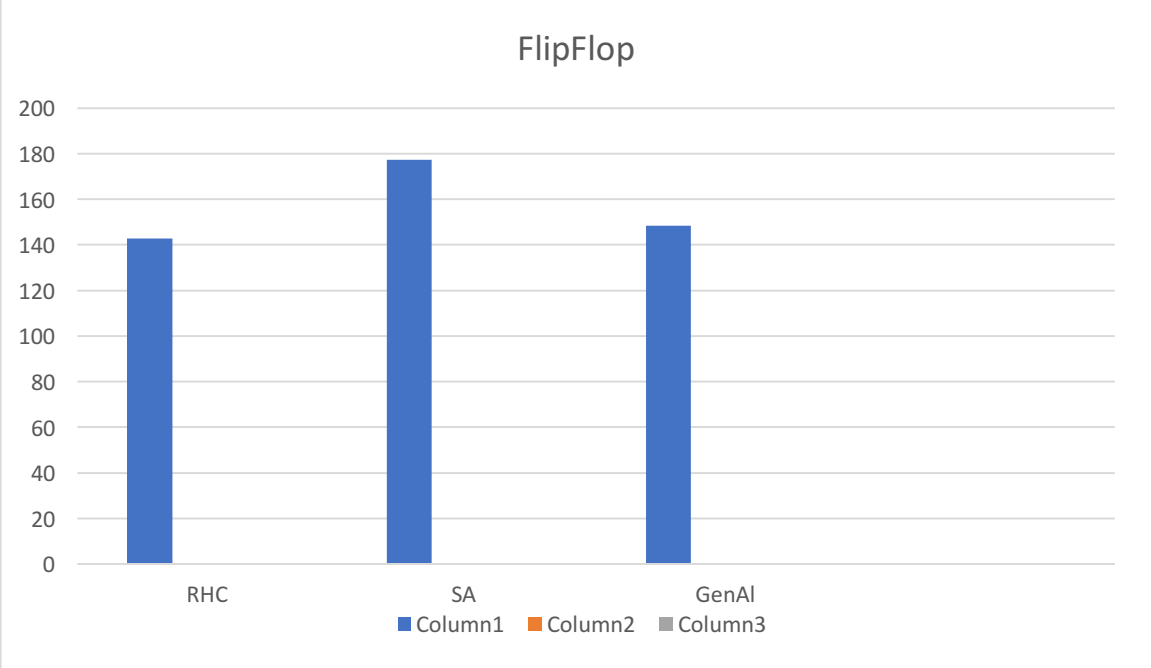
settling on local optima, I would consider it to be the best optimization algorithm for tuning the weights of a neural network.

Part 2: Optimization Problems

Part 2.1 – FlipFlop (SA strengths):

ALGORITHM	AVERAGE OPTIMAL VALUE	AVERAGE TIME (SECONDS)
RHC	142.9	0.5832
SIMULATED ANNEALING	177.2	0.4502
GENETIC ALGORITHM	148.4	0.2021

*SA Temp = 100 CR = .95, GenAl POP = 200 Mate = 100 Mutation = 20



FlipFlop is a problem that return the number of time a bitstring switches from 0 to 1 or 1 to 0. For example, “1010” would return 4, and “11100000000” would return 2. Using this, the optimal solution is a constantly alternating bitstring that would return the length of the string itself. The FlipFlop problem used bitstrings of length 180, so the optimal solution would be 180.

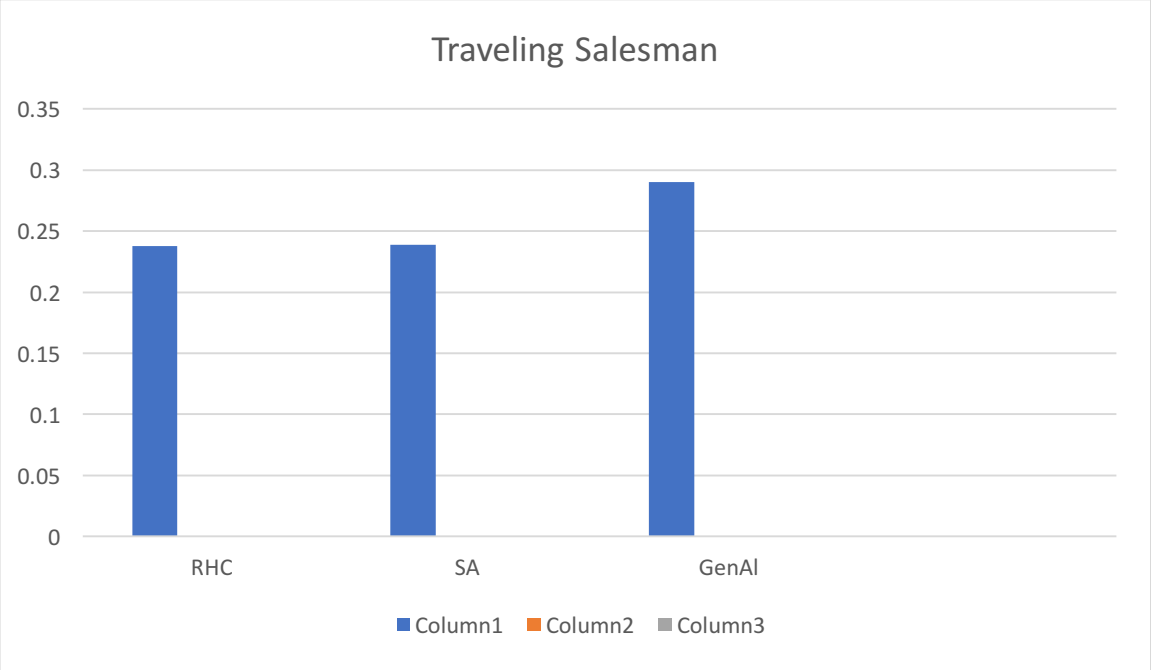
It is clear that SA is by far the most effective algorithm, almost reaching the optimal solution. This is likely because SA works well in spaces where local optima and neighbors are close, so that the temperature can be useful before the cooling rate diminishes it. In this problem, the neighbors are just a flip of one bit in the bitstring. RHC is not as effective because

it could easily reach a string where any changes would not result in a gain in fitness, so it would become stuck in a local optimum. GenAl isn't as effective either because at a certain point, crossing over have an equal chance of decreasing the score by 1 as it does to increase the score by 1, so mutation is the only method for improvement.

Part 2.2 – Traveling Salesman (GenAl strengths):

ALGORITHM	AVERAGE OPTIMAL VALUE	AVERAGE TIME (SECONDS)
RHC	0.237765763	0.0222
SIMULATED ANNEALING	0.238818764	0.1623
GENETIC ALGORITHM	**0.289911567**	**0.1149**

*SA Temp = 1e^12 CR = .95, GenAl POP = 200 Mate = 150 Mutation = 10



Optimization algorithms are ideal for traveling salesman, because it is an NP-hard problem that is much easier to approximate than attempt to solve. The problem is to find the shortest distance a salesman would have to travel to reach all the cities, or to find the shortest route to visit all of the N nodes in a graph.

GenAl performed the best in this problem. In addition to finding the greatest optimal value, it did so in the second fastest time, despite it generally being seen as a slower algorithm. This is true in a complex, computationally intensive problem like tuning weights in a Neural

Network. However, in a comparatively simple problem like this one, the additional amount of time taken compared to RHC is not significant enough to disqualify it as the best algorithm.

In order to find the optimal solution to this problem, it is best to be able to look at larger sets of potential solutions. Changing the route of the salesman has compounding effects. In other words, incrementing a single solution at a time can result in drastic branch of the solution and is, intuitively, not going to be as effective as looking at 200 (Population = 200) solutions and using the higher performing members of that set to create new members.

Part 2.3 – Conclusion:

We see in the FlipFlop problem that when there are many local optima, RHC is not able to find the optimal solution effectively, and when the crossing over two solutions has an equal chance of hurting than it does helping, GenAl's strategy falls apart. SA excels in overcoming local optima, and therefore performs the best in FlipFlop.

The traveling salesman problem reveals a scenario in which iteratively incrementing a single solution point is not as effective as simultaneously utilizing numerous solutions. It also shows that GenAl isn't always a time-consuming algorithm, rather it depends on the problem being applied and the hyperparameters being fit.

Works Cited:

Code: https://github.com/james7132/GTCourseWork/tree/master/CS%204641%20-%20Machine%20Learning/Randomized%20Optimization%20Assignment

Dataset: https://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+(Version+1)