Zane Motiwala

Machine Learning Fall 2018

Assignment 2 – Randomized Optimization

Part 1: Use Random Search Optimization algorithms to train NN

Implement four local random search algorithms and use the first 3 instead of backprop to train a neural network on a dataset from Assignment 1

- RHC: Randomized Hill Climbing
- SA: Simulated Annealing
- GA: Genetic Algorithm
- MIMIC: Mutual-Information-Maximizing Input Clustering

Answer/Analysis:

The first step was to implement the four random search algorithms. After several hours of trial and error I finally figured out how to use the ABAGAIL coding resource. Once that was operating the rest of the coding was trivial. I was able to run the command line code from the Wiki to run 4 algorithms. For example, I ran the XORTest.java and TravelingSalesmanTest.java.

For the next part, I used the breast cancer dataset that I used from Assignment 1. I essentially copied the same structure as the AbaloneTest file except I modified the code to read in my dataset and identify the correct attributes and classification column. I also added another hidden layer to ensure the parameters matched my optimal results from Assignment 1 which had an accuracy of 91%. Below are the results of the 3 algorithms we were supposed to use to train Neural Networks instead of backprop at 100, 1000 and 5000 iterations:

100 iterations

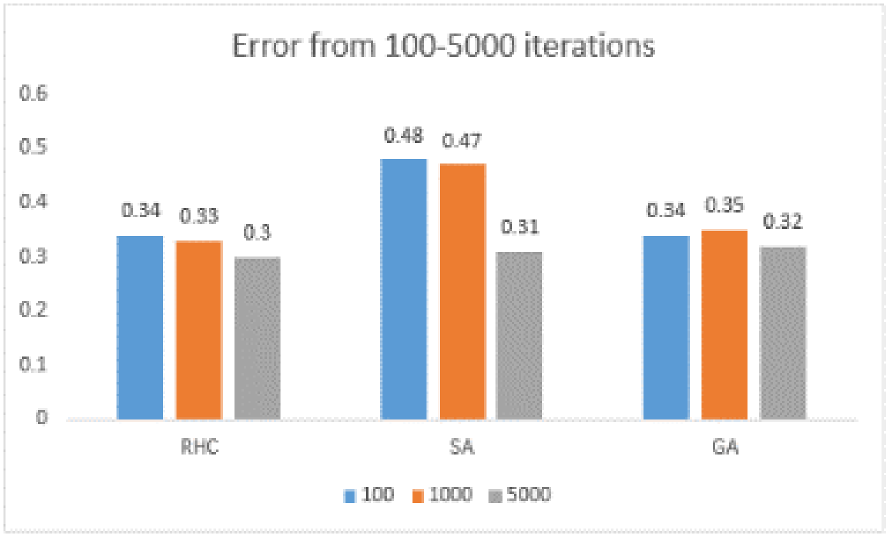
Algorithm	Correctly Classified	Incorrectly Classified	Error %	Training Time	Testing Time
RHC	503	265	0.34505	2.914 s	.021 s
SA	398	370	0.48177	3.110 s	.018 s
GA	506	262	0.34115	129.627 s	.019 s

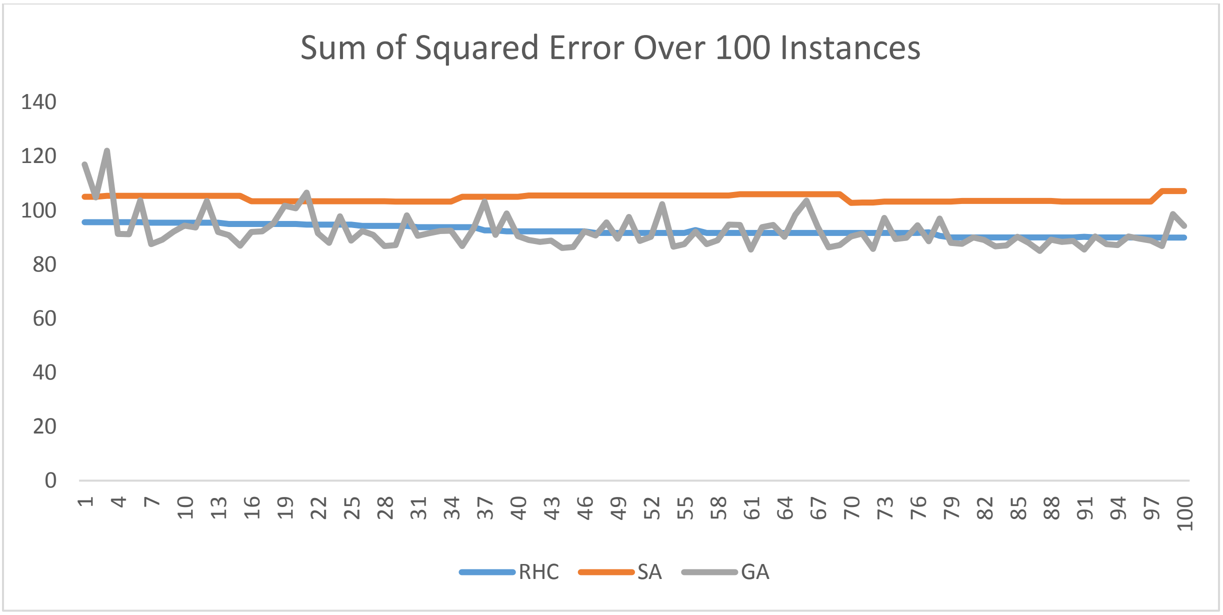
1,000 iterations

Algorithm	Correctly Classified	Incorrectly Classified	Error %	Training Time	Testing Time
RHC	517	251	0.32682	18.068 s	.011 s
SA	404	364	0.4736	17.974 s	.012 s
GA	493	275	0.35807	936.748 s	.012 s

5,000 iterations

Algorithm	Correctly Classified	Incorrectly Classified	Error %	Training Time	Testing Time
RHC	534	234	0.30487	92.816 s	.012 s
SA	531	237	0.30859	89.719 s	.011 s
GA	526	242	0.3151	5794.537 s	.019 s





At first I didn't understand the benefit of this assignment until I started researching online that backpropagation may not be the best at finding the global solution and may select local solutions. I learned that the vast majority of neural network research relies on a gradient algorithm, typically a variation of backpropagation, to obtain the weights of the model which is susceptible to stopping at a local maximum. Therefore, it would be interesting to see if using these random search techniques which are able to find global maxima would be better instead of backprop and would provide more accurate and consistent results. Though my results don't indicate anything better than backprop (at least with these 3 on this dataset), below is some more analysis on each technique that was tested.

- **Random Hill Climbing:** is a searching technique that looks for the global optimum by moving towards the next higher elevation neighbor until it reaches the peak. This algorithm randomizes its original starting position to help eliminate getting stuck at a local optimum and increases the probability to choose a restarting position that will bring it closer to the global optimum. This should work well in case of neural networks as it works on a similar principal of gradient descent to find the optima, but will randomly restart and test more optimums. One of the pros of this algorithm is that it is easy to implement and manipulate. If there were only a few optimums and the data wasn't too skewed this technique would be very useful and fast. However, it still doesn't guarantee any optimal solution especially when there are many optimums or if the data is skewed and you happen to not fall in any of those areas with the optimum. In these cases you are still left to chance when trying to find the optimal solution. It seems there could be more information gained if the restarts were not just random, but based on some probability. I guess that is why Professor Isbell wanted to retain the structure and develop an algorithm like MIMIC.

- **Simulated Annealing:** is inspired by metallurgy where metals are heated to high temperature and then slowly cooled to increase ductility. The algorithm considers a neighboring state of the current state, and probabilistically decides the step towards the next neighbor. Compared to hill climbing the main difference is that SA allows downwards steps. Simulated annealing also differs from hill climbing in that a move is selected at random and its acceptance is conditional. Yet, moves that improve the cost function are always accepted. It uses the temperature T to define how randomly to select and the algorithms starts off with high T to allow for more random search in the beginning and slowly cools off to take less random steps. Therefore, as the temperature decreases, the probability of accepting worse moves decreases. This iterative process is repeated until the system reaches a state that is the optimal or a given computation budget is exhausted. By starting off with more randomness and wider search the algorithms can explore more area to find the global maximum and not get stuck in local maximums. I was very surprised that this algorithm had higher error than RHC. I ran it for 1000 iterations and still didn't get the best performance. It wasn't until I ran it for 5000 iterations that I was able to get similar results to the other 2 algorithms. If I had to test again I would play with the parameters T and the number of iterations at each T to get better performance. I don't see how RHC could outperform SA if SA was optimized properly. One of the things I learned is that the benefits of this algorithm can be seen with slow cooling and lots of iterations.

- **Genetic Algorithm:** provides solutions to the optimization problems using techniques inspired by natural evolution such as reproduction, mutation, and crossover. These algorithms initially start from a population of candidate solutions and continuously evolve to a better solution after a number of iterations by eliminating non adequate sections of the population and retaining the best parts of the population. The best traits can be determined by mutating and mating different parts of the population. The concept is easy to understand, and performs well in “noisy” environments. It can be applied when the objective function is not smooth where derivative methods cannot be applied. Some disadvantages are that you usually need a decent sized population and a lot of generations before you see good results. One of the other difficulties is trying to fine tune all the parameters. There are so many options that you need lots of trial and error to find the optimum of the optimum GA algorithm. I was surprised to see the GA algorithm perform the best in terms of error rate on the breast cancer dataset. It seemed as if it zig-zagged (mutated/crossed-over) between the results of RHC and SA and ended up the best. I tested up to 1000 iterations and GA still was a little better. It seemed it was able to go beyond just the training data and essentially create its own better samples and obtain better results. I was really impressed with GA, however, it took much much longer and would not be scalable for any real-time application.

Overall, the genetic algorithm had the lowest error, but took much longer. It took over an hour to complete 5,000 iterations, but it required less iterations to converge and get high accuracy. I was surprised SA didn't perform better at 1,000 iterations as in theory I believed that it would perform better than RHC at all iterations. If I had more time I would play around with the Temperature setting and number if iterations to figure out how to get better performance with SA. I am interested to seeing how it performs on the 3 optimization problems below. Nothing came close to the accuracy of backpropogation though intuitively there is still good reason to believe that improvements can be made since backprop is not perfect and there are pros to these random search algorithms. If I had more time I would try different problems and see if these 3 algorithms performed better.

Part 2: Analyze 3 Optimization Problems

Create 3 optimization problems (maximize a fitness function) and apply all 4 search techniques (RHC, SA, GA, MIMIC). The first problem should highlight the advantages of GA, the second SA, and the third MIMIC.

Answer/Analysis:

I tested all of the optimization problems that were in ABAGAIL (see table at the bottom). I ended up picking the following three to highlight the advantages of GA, SA and MIMIC.

(GA) Traveling Salesman

- Overview: This is a really popular problem. It asks "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" This problem is important because the amount of routes is calculated as a factorial of the number of locations. If we want to find the shortest route for our map of 20 locations we would have to evaluate 2432902008176640000 different routes.
- Results: At first I didn't think GA would be a good option for this problem, but it ended up being the best. Though the methods/parameters need to be tuned properly since adding locations drastically increases the number of routes, we would not want the GA to mutate or reproduce and create new locations. Rather, just swap mutations and not increase any locations. **Why is GA the best**? When you think about it, if I were to calculate which route would be best that is essentially how I would do it. Swap the order of routes and test how long they take. That is the basic functionality built into the GA algorithms which can mutate and swap the orders around and that is why I think it worked so well on this problem. Similarly, when thinking of Parent and Offspring, the routes can keep half from the first instance and take the second half of locations from the next instance and test. This helps preserve the ordering of multiple locations within each iteration and the algorithm can track whenever any element increases the time it takes to proceed on a given route.

(SA) Continuous Peaks

- Overview: The continuous peaks problem is a continuation of the 4 peaks problem and is a good example in figuring out the highest peak vs lower peaks (global maxima vs local maxima). The function counts the longest run of ones and longest run of zeros anywhere in the bit string.
- Results: Simulated Annealing performed the best on this problem. MIMIC also did well, but as the number of iterations grew SA performed better and faster. **Why?** Intuitively, this highlights the main advantage of SA since it has the ability to change the amount of randomization as the algorithms learns. In the beginning the temperature is high so the random search will be able to explore a wide variety of the "peaks" and slowly be able to minimize the randomization and converge to the global maximum across the peaks. Going back to the heating and cooling of metals analogy and how that straightens and aligns the particles in metals, in this problem we are looking for a continued streak of 1s or 0s and so I think that same analogy is why this algorithms performs the best.

(MIMIC) Knapsack

- Overview: I thought this was a really interesting problem that occurs in everyday life. I didn't know I could write an algorithm for such a problem, but am impressed to see all the algorithms try to figure out the combined optimization for multiple items in a bag. The setup: For n elements, each with a mass and a value, 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. Backpackers face this problem when given a fixed-size knapsack and have to fill it with the most valuable items for their hike.
- Results: MIMIC outperformed all other algorithms, though it took more time. **Why?** Intuitively, this doesn't seem like a problem where pure random actions or mutations will be of benefit unless a fairly large number of iterations are used. It seems like knowledge of a situation and the structure of the problem will be of use in every iteration. A hiker knows what is essential and what can be less essential. Therefore, this seems like a perfect fit for MIMIC. This may have to do with the structure of the learned distribution. I read online that Knapsack problems are frequently solved with dynamic programming, and the strength of the dynamic programming approach is that, when broken into sequential stages, the optimal solution given the current stage must include the optimal solution of the next stage. Solutions are worked backwards by selecting the optimal decision given all prior decisions. MIMIC is a similar process, working backwards from solutions, iteratively recalculating a distribution from the most valuable generated solutions, each variable conditioned on the value taken by the parent variable. The learned dependency tree represents a sequence of decisions of how many of each item to include at each stage, and each path taken through the tree resembles stages of a dynamic program. Most of the optimization algorithms lack structure and MIMIC uses knowledge of this structure as a guide for randomized search through the solution space. It attempts to communicate information about the cost function obtained from one iteration of the search to later iterations of the search directly.

Fitness Function Evaluations on Optimization Problems

Algo	FourPeaks	ContinuousPeaks	CountOnes	FlipFlop	Knapsack	NQueens	TravelingSalesman	TwoColors
RHC	**200**	60	71	65	2545	**44**	0.135	119
SA	**200**	**113**	54	**79**	2491	**44**	0.1186	98
GA	19	91	50	68	3028	**44**	**0.159**	116
MIMIC	80	112	**79**	77	**3102**	42	0.0999	**189**