Analyzing Randomized Optimization Problems

Srinidhi Palwayi

Introduction:

In the first part of the paper I will analyze and show how randomized optimization algorithms, particularly, randomized hill climbing, simulated annealing, and genetic algorithms performed on finding weights for a feedforward neural network. This is the same type of neural network that was used on the first project, but the first project's neural network was trained using back propagation. In addition, in my first project I got the best precision and recall scores when I used a linear activation, so for this project I will used a linear activation for my experiments. Furthermore, I will compare the results that I got from my first Project using the Wisconsin Breast Cancer Dataset with the results that I get from using randomized optimization methods. When discussing precision and recall as a function of the number of training examples, I'm comparing my randomized optimization results to Figure 1, which shows the scores when using back propagation.

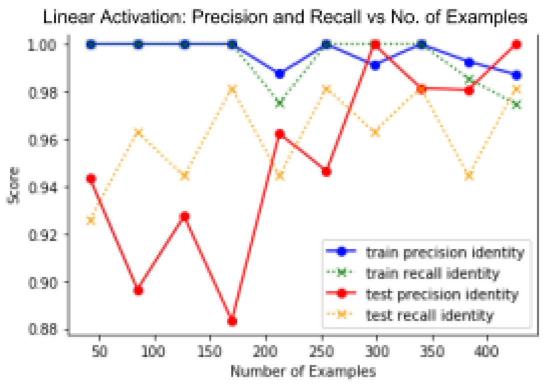


Figure 1

In the second part of the paper I will discuss problems that I find interesting and how each problem is more fit for either simulated annealing, genetic algorithms, or MIMIC.

Section 1: Randomized Optimization on Neural Network Weights:

Randomized Hill Climbing (RHC):

Randomized hill climbing, works by sampling the neighbors of the current solution that you are exploring, and moving towards the neighbor that has a better fitness. The initial distribution of the weights starts of as uniformly random, and then as the algorithm runs better weights are found.

In Figure 2, we can see that as the number of iterations increase, the precision and recall scores start to converge and become less prone to large dips in score. This makes sense because as we increase the number of iterations we are making it more likely to find the optimal weights. Here we see that random hill climbing converges around 300 iterations. This is interestingly, about 3 times as many epochs it took took for back propagation to converge. However, epochs aren't directly comparable to iterations, because an epoch is a forward pass and a backward pass through the neural network. Furthermore, I used batch gradient descent, which means that I updated the weights after every batch, and that every epochs had 3 batches. Hence, when using back propagation as well there was 300 iterations to get the model to properly fit. Lastly, the test precision scores were higher and more consistent for the back propagation network compared to the random hill climbing network.

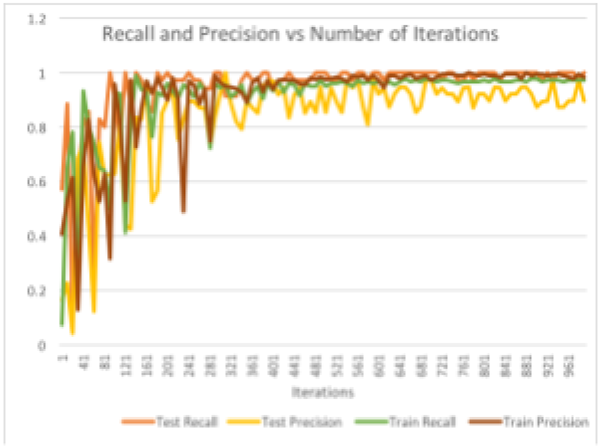


Figure 2

In Figure 3, I measured precision and recall as a function of the training examples, and kept the number of iterations at 300. The only clear improvement comes from the precision score as we increase the amount of training examples. Interestingly, in Figure 1 as well we can see that precision score was the most improved when using backpropagation to find the weights. Furthermore, test precision performed about 10 percentage points better when using back propagation.

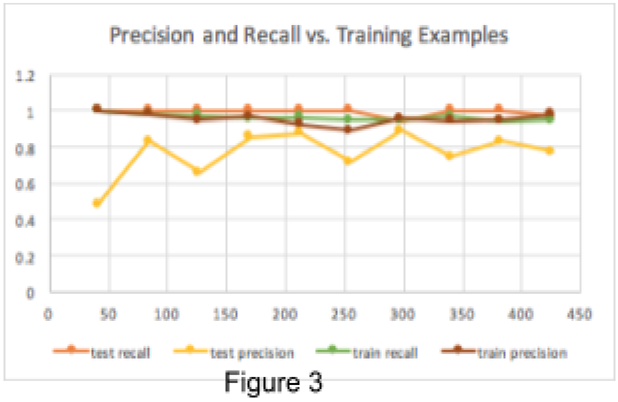


Figure 3

Simulated Annealing (SA):

Simulated Annealing is very similar to RHC, but SA sometimes explores what seems like a worse solution. By exploring a worse solution, we have a better chance of finding the global optima, rather than getting stuck at a local optima. The likelihood of exploring what seems like a worse solution is given by the temperature, which starts off high and then cools down. This means that we are more likely to explore what currently seems like a worse solution when the temperature is high, but then as the temperature cools we are less likely to take a downward step.

The cooling rate is how quickly the temperature cools down, and in Figure 4, it shows how the precision and recall change as the temperature cools down at a slower rate. It's interesting to note that between 0.55 and and 0.7 cooling rate, recall and precision scores are better or equally good as RHC when using 200 iterations. However after a cooling rate of 0.75 the score goes down significantly, which shows that the cooling rate is too slow to find a optimal in 200 iterations. For this experiment the temperature started off at 10.

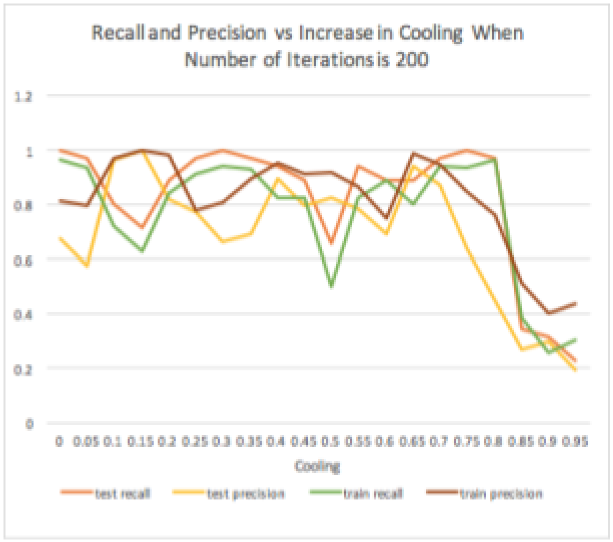
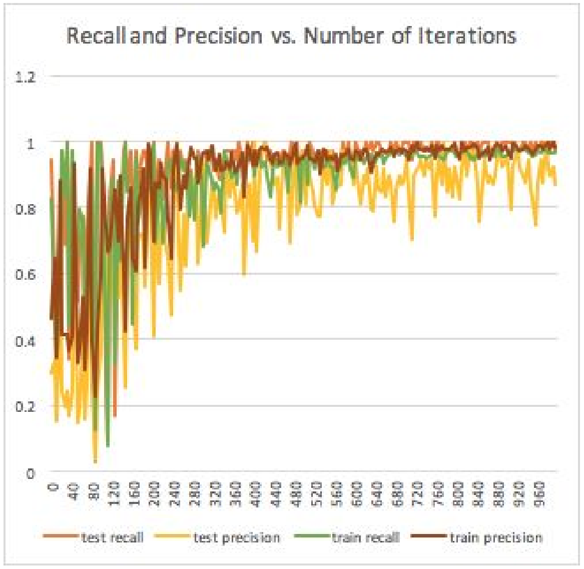
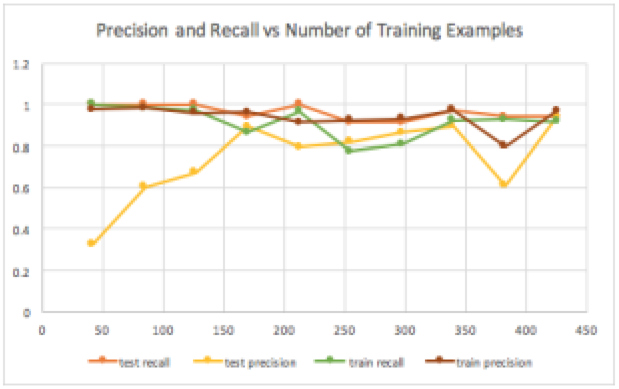


Figure 4

In the experiment below I modified the number of iterations and measured recall and precision. The test recall, train recall, and train precision, converged about a 100 iterations after RHC. Furthermore, the test precision for SA has more variance than the test precision for RHC.



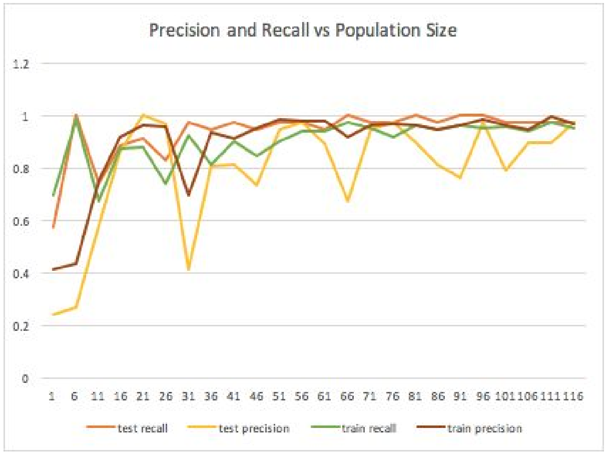
Another experiment that I performed, was seeing how the precision and recall varied with the amount of training examples. For the experiment below, I kept the number of iterations at 300 and the cooling rate 0.55. Again, similar to when using back propagation and randomized hill climbing the most significant improvement came from test precision. This also shows that recall generalizes pretty well with only 10% of the dataset, meaning that a neural network is doing a good job at making sure that tumor doesn't go undetected.



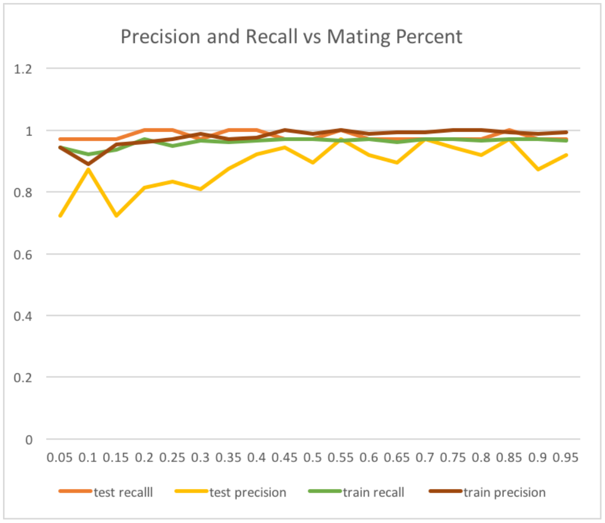
Genetic Algorithms (GA):

Genetic Algorithms are an analogy to biology and how populations evolve. The population are a set of possible weights for the network, and then the fittest individuals are selected to mate or mutate to create the next population. For all the experiments below I kept the number of iterations at 200, keep the population at 120, mating at 30% of the population, and mutation at 10% of the population, unless otherwise noted.

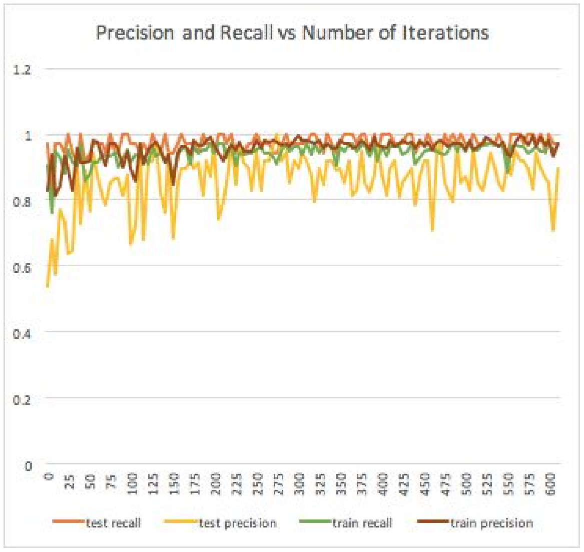
In my first experiment I changed the population size as seen in the figure below. In this figure it's interesting to note that the scores tend to increase as the population size increases. Furthemore, variance between nearby population sizes decreases as the size of the population increases, similar to how in RHC the graph is less volatile as the number of iterations increases. This makes sense because as we increase the population size, it's more likely that the optimal combination is in the population. Furthermore, it shows the importance of having a diverse population in arriving at the optimal solution.



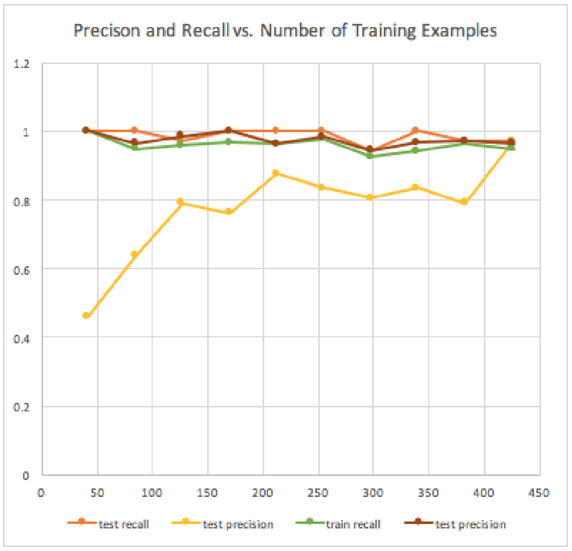
In my second experiment I changed the mating rate of the population. The score that changes the most is test precision. This shows that crossover best benefits precision in creating more fit children. Furthermore, the “mating” doesn’t seem to affect recall much because recall already starts of very high.



Next I experimented with the number of iterations. Compared to to the number of iterations graph in RHC, at lower iterations we see that genetic algorithms can generalize better and score better. In addition, the most variation in scores came from precision in both the graphs. However, unlike the RHC number of iterations graph, the variation in test precision doesn’t seem to decrease with increasing number of iterations. But, for the remaining scores, the variation does decrease.



Lastly, as the number of training examples increased, the precision showed the most amount of improvement. Again, showing that increasing training examples, benefits test precision the most, and helps mitigate overfitting. Lastly, a low precision means that there’s a large amount of false positives. Hence, there’s an especially large amount of false positives before 200 training examples.



Comparisons:

Overall, the best randomized algorithm for the breast cancer dataset is genetic algorithms. This is because it had the best testing performance on precision and recall. Furthermore, RHC had comparable test recall and precision, but it was more prone to dips when measuring the number against number of training examples. Lastly, the randomized optimization algorithms performed comparably to back propagation on almost all scores, but test precision was consistently able to do better with fewer training examples using back propagation. This relative similarity between randomized optimization and back propagation might occur due to the relatively small network with only 3 hidden layers with 13 nodes each. Hence, it may be easier to find random optimal weights.

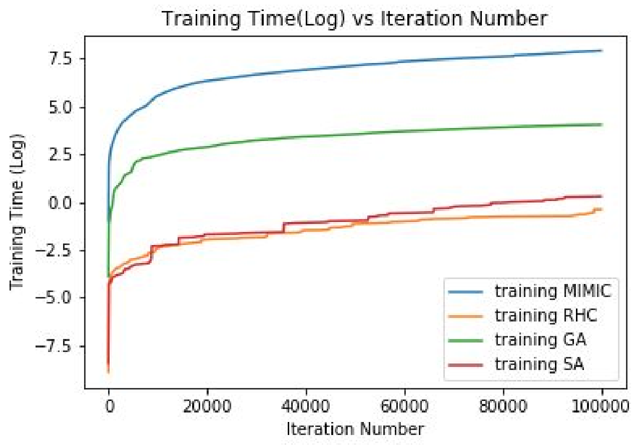
Section 2: Interesting Problems

Four Peaks (MIMIC):

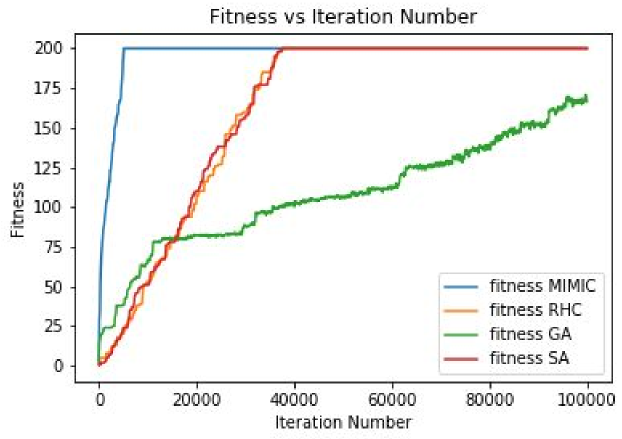
The four peaks problem has two local maxima, and two global maxima. On one hand, the local maxima is created by having a bit strings that consists of entirely of 1's or entirely of 0's. On the other hand, the global maxima is created by having greater than a certain threshold, T, 1's followed by T+1 0's, or T+1 0's followed by T+1 1's. When the value of T becomes larger the problem becomes more difficult, because the basins of attraction for the local minima grow larger as well.

For this simulation the bit string was length 200, and T was 40. Additionally, for simulated annealing the temperature was 1E11 and the cooling rate was 0.95. For genetic algorithms, the population, mating rate, and mutation rate were 200, 0.5, 0.1, respectively. Lastly, for MIMIC, the sample size was 200, of which only 20 were kept in each iteration.

As seen in the figure below, RHC takes the least amount of time because it's the least complicated algorithm. The longest training time was seen for MIMIC. This makes sense because each iteration of MIMIC is costly because at each iteration we reap a lot more information by creating things likely dependency trees and maximal spanning trees.



Interestingly, genetic algorithms never found the optimal solution to the four peaks problem. This makes sense because the optimality of the parent doesn't necessarily guarantee that the children are going to be more fit since each bit isn't independent. In the graph below we can see that MIMIC has the fewest amount of iterations to find the optimal solution. MIMIC is the best algorithm for the four peaks problem because MIMIC was able to exploit the underlying structure of the problem.

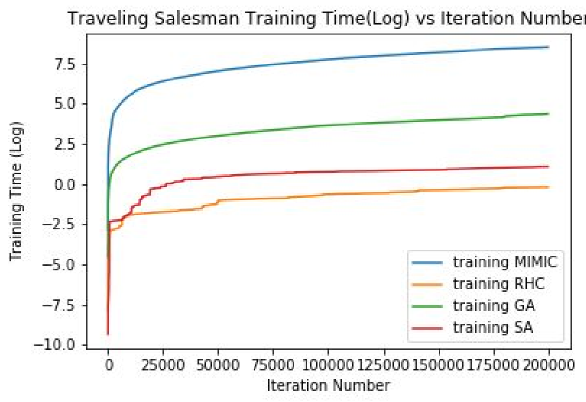


Traveling Salesman (Genetic Algorithm):

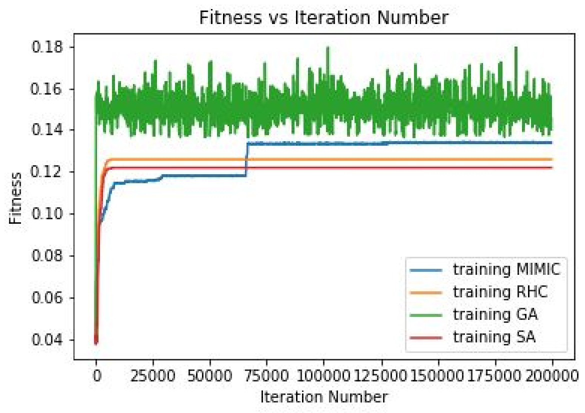
The premise of this problem is that a salesman wants to travel between a number of cities and return back to the origin, but wants to do it on the shortest possible route.

For the experiments below simulated annealing used a temperature of 1E12 and cooling rate of 0.95. Genetic algorithms had a population of 200, mating rate of 0.75, and mutation rate of 0.1. Lastly, MIMC has 200 samples, of which 100 are kept in each iteration.

Similar to the Four Peaks time graph, we see that MIMIC takes the longest time, followed by GA, SA, and RHC.



Genetic Algorithms performed the best on this problem. Although it didn't get the most optimal solution every run, as seen by the jagged line, but it did always perform better than MIMIC, RHC, and SA. Interestingly, MIMIC performs less well than RHC and SA until about about 7000 iterations. Furthermore, it seems that MIMIC is getting stuck at local optima.

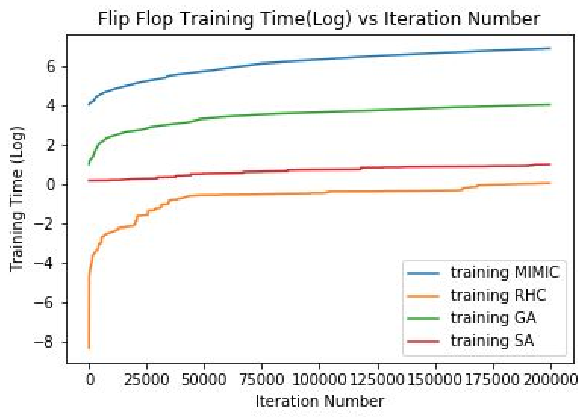


Genetic algorithms performs the best on this problem because the structure of the problem allows fit parents to create fit children. Hence it takes advantage of crossover when mating.

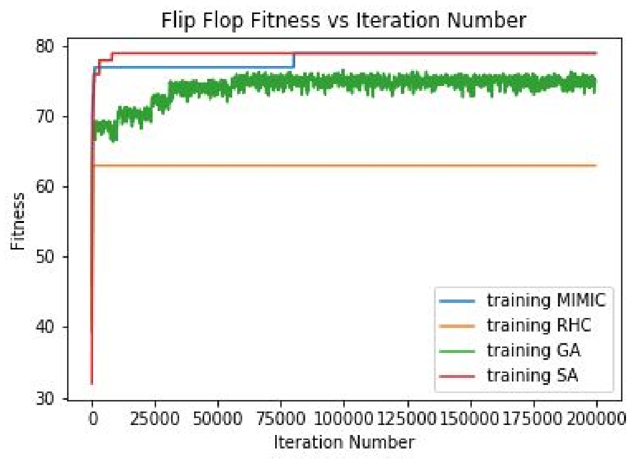
Flip Flop (Simulated Annealing):

The goal of this problem is to find the number of alternating bits in a given bit string. Hence, the best solution will be a bit string with all alternating bits.

For the experiments below simulated annealing used a temperature of 100 and cooling rate of 0.95. Genetic algorithms used a 200 population size, 0.5 mate rate, 0.05 mutation rate. Lastly, MIMIC used 200 samples, and kep 20 in each iteration.



For this problem the best algorithm was simulated annealing because it took the shortest number of iterations to achieve the optimal solution. Furthermore, it seems that by sometimes following what seems like the less optimal solution, SA reaches the global maxima, while RHC seems to get stuck in local optima. Genetic algorithms don't work well in this situation because the optimality of the bit string is not feature independent, meaning the best choice for the current bit depends on the previous bits. Hence, combining optimal parents, won't necessarily increase the fitness of the children. Lastly, MIMIC doesn't seem to be able to exploit the structure of the bit string to produce optimal results until about 4 times the number of iterations after simulated annealing has reached the optima.



Conclusion:

Overall, randomized optimization is very helpful over supervised learning methods in situations when there's a complex underlying function, no derivative to the function, or many local optima. Furthermore, it's important to recognize the domain knowledge that is being used when applying these algorithms. For instance, for simulated annealing the assumption is that with enough steps down or restarts we can eventually find the global optima, and not get stuck in basins of attraction with the local optima. For genetic algorithms the assumption is that each dimension of the subspace matters independently and that fit parents produce fit children. For MIMIC we are assuming that the function has structure that can be modeled. In addition, MIMIC fixes two key problems of the other randomized optimization algorithms we discussed. First, we can't capture the history of the past iterations of the algorithm. Second, we can't capture the structure and probability distribution of the points. Hence, when looking at problems, like four peaks, we find the solution in much fewer iterations.