Andrey Lukin
CS 4641 Machine Learning
Project 2: Randomized Optimization
Professor Charles Isbell

Project 2: Randomized Optimization

Introduction

For Project 2 in CS 4/7641 we are shifting from the normal back propagation technique used in the majority of neural networks, to finding other algorithms that can be used to find the weights for the neural network. In this project we will be comparing how four different algorithms preforms, with the four algorithms being Random Hill Climbing, Simulated Annealing, Genetic Algorithm and MIMIC. This project is broken into two parts, the first testing how the first 3 algorithms perform with a dataset from Project 1, and the second part will be testing all four algorithms on three optimization problems to see how they perform. For the first part of this assignment, I'll be using the Titanic dataset which contains details regarding passengers of the famous Titanic ship, and I will be testing whether they survived the iceberg crash or not. For the second part of the assignment, I will be using the Knapsack algorithm, Travelling Salesman, and N-Queens. I'll be using the open-source Java library ABIGAIL for the implementations of all the algorithms.

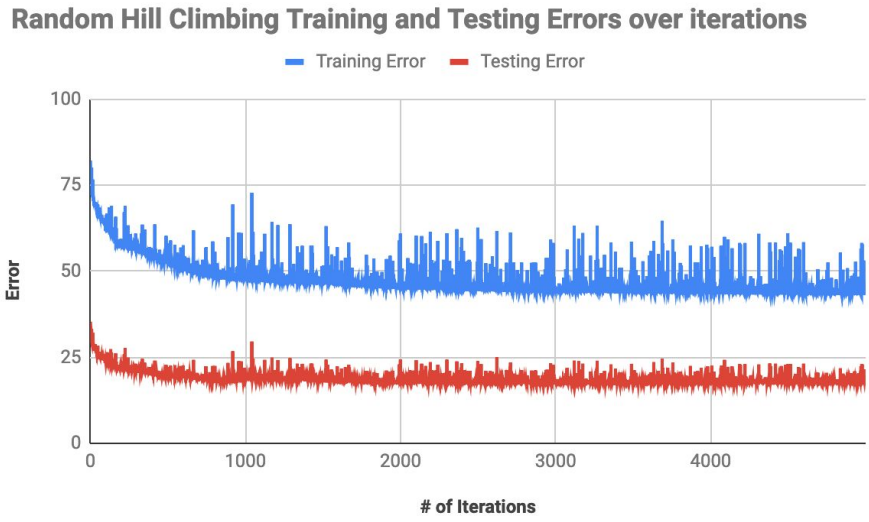
Part 1

Here we will be comparing the overall performance of using Random Hill Climbing, Simulated Annealing, or Genetic Algorithm to find weights for our nodes in the neural network. We will be using the Titanic dataset, where we will be using the class, sex, ticket fare, the location of embarkment, whether sibling/spouse, whether parent/child of a person to predict whether or not they survived the Titanic crash. We are running every algorithm with 5 hidden layers and for 5000 iterations

Random Hill Climbing

Random Hill Climbing is an algorithm that uses the neighbors of the point in question to decide which way to move to find the overall maxima of a function. This maxima is then used to find weight for the NN, which is what we want!

One potential problem of the Random Hill Climbing algorithm is the potential of the algorithm to get stuck on a local maxima. If a local maxima is reached, the neighbors have lower values than the current value, and so the algorithm thinks it reached the best maxima, even though it's only local. To combat this, there is the idea of random restarts in Random Hill Climbing: when a maxima is reached, restart the algorithm multiple times and see if you can find anything higher than what was already reached. This is the only hyper parameter for Random Hill Climbing, the number of restarts. Unfortunately, ABIGAIL's implementation of Random Hill doesn't easily allow the additions of restarts



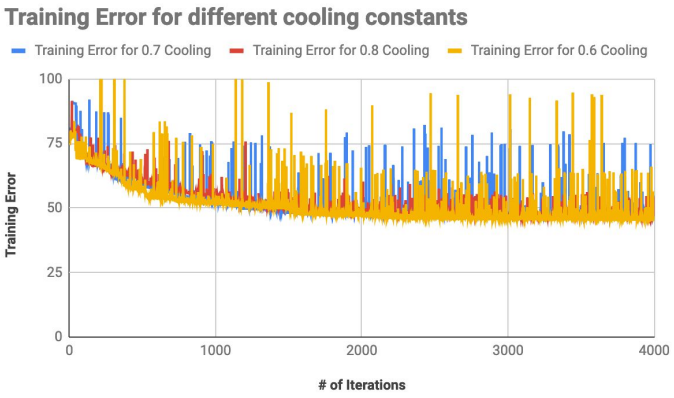
Here we see that the testing error converges at about 700 iterations at an error at about 19%. Even though that might seem like a large number for the number of iterations, one positive thing about this algorithm is its simplicity. This can be run quickly and get results quickly.

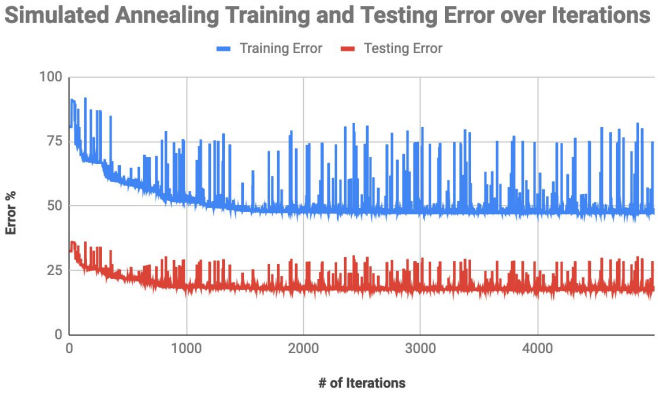
Simulated Annealing

Simulated Annealing uses a similar philosophy as Random Hill Climbing: it uses the neighbors of the current point to decide which way to move to find the universal maxima. But unlike Random Hill Climbing, there is a depleting chance of a “random restart” based on the formula here. As time progresses, the lower the chance of there being a “smart” random restart. Based on the formula,

$$P(x, x_i, T) = \begin{cases} 1 & \text{if } f(x_i) \geq f(x), \\ e^{\frac{f(x_i)-f(x)}{T}} & \text{otherwise} \end{cases}$$

two things to keep in mind would be that as we lower the initial temperature to 0, it becomes



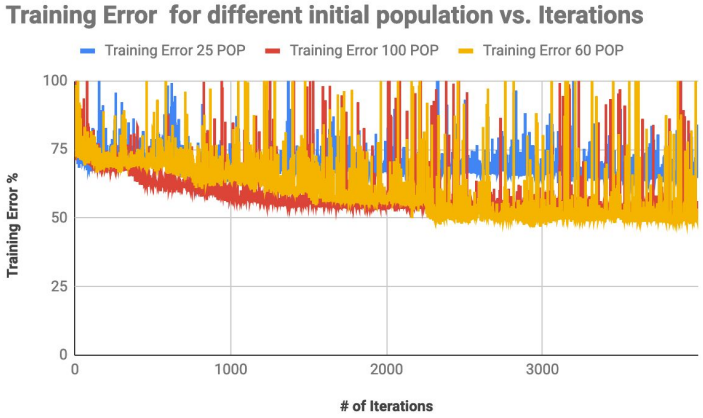


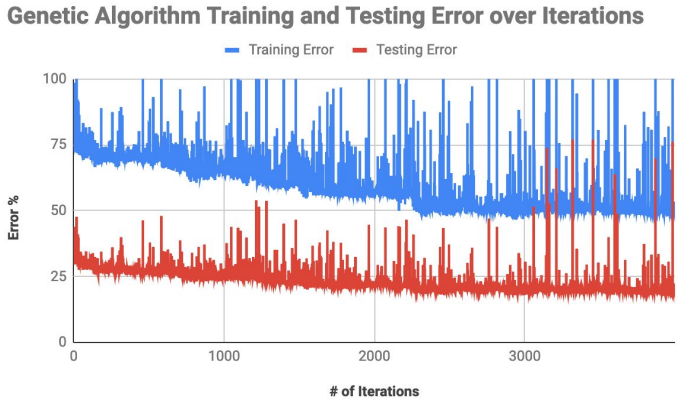
more like Random Hill Climbing, but as we increase the temperature to infinity it starts caring less about the neighbors and is more likely to do random restarts.

The two hyperparameters for Simulated Annealing are the starting temperature and the cooling factor. The graph above was the test to see what would be the best cooling factor for the data, and since there weren't noticible differences, I ran the final algorithm with a cooling constant of 0.7. The starting temperature was decided separately, but an initial temperature of 100 had the least overall error with testing. It is seem that after around 1000 iterations, the testing error stays at the 19% error.

Genetic Algorithm

Genetic algorithms try to simulate the process of evolution, where the most "fit" survive and are able to reproduce, which allows them to be able to pass the trait that made them more "fit" to survive. Additionally, the thing that helps variability, are mutations: a form of random chance adding the possibility of an offspring to acquire traits not necessarily possessed by their parents.For this algorithm, there are three hyperparameters: the size of the initial population, the number of mating that happens per iteration, and the number of mutations per iteration.





Here I tested what would be the best initial population. I tested a high and a low, and one in the middle with the values of 25, 100, and 60 respectively. As seen on the graph, 60 overtakes 100 after about 2300 iterations, and since we are using 4000 interactions for all of our tests, we will be using 60 as the initial population. For this assignment, similar tests were run for number of mates per iteration and number of mutations per iteration, and the values of 25 and 10 respectively did the best with the data.

Here we see that the testing error is continuously decreasing, even if very slowly, but it seems to plateau at about 2500 iterations and at 19% testing error.

Conclusion of Part 1

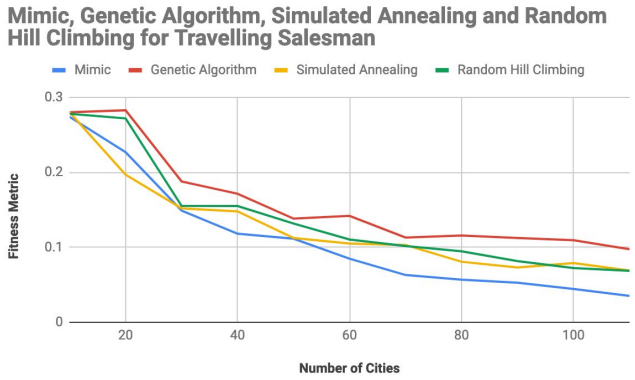
Even though all 3 algorithms preforms basically the same, I think that Simulated Annealing would be the best out these to use with a neural network. The reason behind this is as follows: Random Hill Climbing, even though it is fast, very much is based on chance of whether or not you get stuck at a local maxima, and the genetic algorithm takes a lot longer to run. To add to the Random Hill Climbing, the Simulated Annealing algorithm is build on top of the algorithm, and is obviously better since with the temperature function we are able to have not just pure randomness, but more controlled variability. With the Genetic Algorithm, the process took a lot longer to run timewise, and even then the testing error converges a lot later. Based on the “No Free Lunch Theorem”, we need to see potential setback and wins, and with all of these factors, Simulated Annealing seems to be the best option.

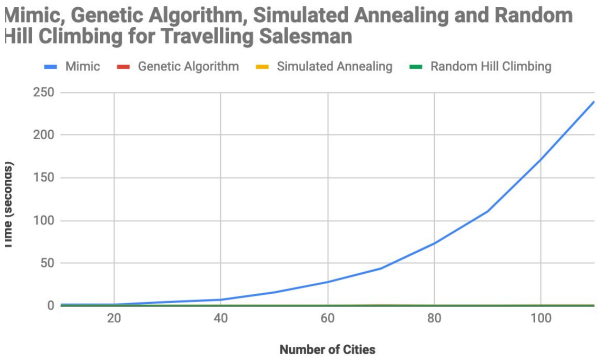
Part 2

For Part 2 of this assignment, I picked 3 optimization problems and am running Random Hill Climbing, Simulated Annealing, Genetic Algorithm, and MIMIC to try finding the maximized solution. For the purpose of this assignment, I will be working on optimizing the Travelling Salesman problem, the N-Queens problem, and the Knapsack Problem.

Travelling Salesman

The Travelling Salesman is defined as follows: given a list of cities and the the distances from one city to the next, what is the most optimal route to take to visit every city and return to the original city you left from. This is an NP-complete problem, and thus brings a lot of interest from anywhere to be solved, and especially with Machine Learning, since it might allow us to save time.





Here are two graphs, one where all 4 algorithms were run with an increasing number of cities, and the performance metric measured, and the second is the amount of time in seconds required to run the algorithm every iteration. In this situation, the performance metric decreases as the model runs better.

Before we discuss the conclusion of the things found in this study, the hyperparameters used for the tests were:

- Simulated Annealing: starting temperature was 10^12, and cooling was 0.95, ran for 200,000 iterations.
- Randomized Hill Climbing ran for 200,000 iterations
- Genetic had a population of 200, 150 mates, and 20 mutations, ran for 1000 iterations.

- MIMIC had 200 samples and ran 100 of them, ran for 1000 iterations

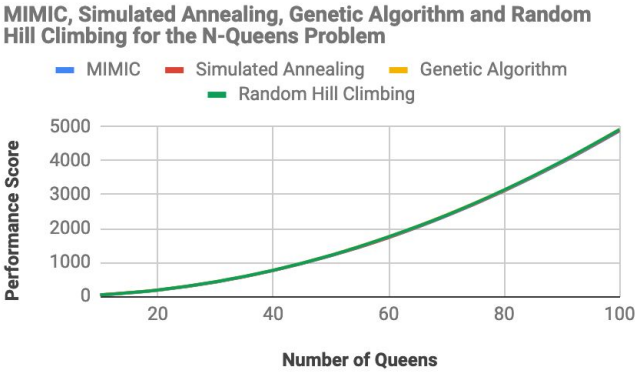
Here we see that the genetic algorithm performed the best. A potential reason why this is the case is that the genetic algorithm is able to break down tasks and them put them back together for the final model. Additionally, since the problem is NP-complete, the fact that Genetics algorithms work based on a congregation of approximate numbers, and works more like a biological simulation allows it to work best in this situation. Interestingly enough, I would think that the time it would take for the Genetics algorithm would be disproportionately big, but it not.

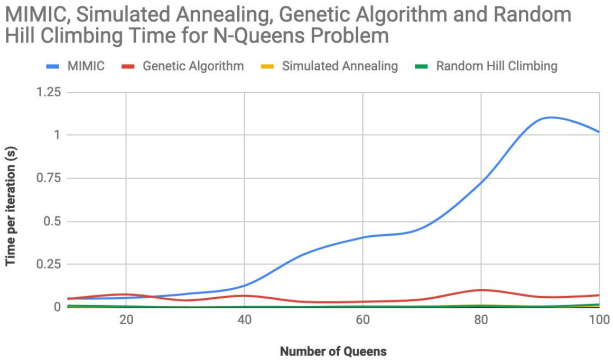
N-Queens

The N-Queens problem goes as such: given an NxN chess board, position N queens on the board so that no other queen can take it. As a small reminder, Queens can move diagonally and up/down left/right which makes them the most valuable piece (after the King of course!). One reason this problem was interesting for me is the testing of the simulated annealing algorithm: in an N-Queen problem there would be no such things like “local maxima” or “local minima”, since you can either put the queens down or not. There isn’t just one solution for every board (just by turning the board 90 degrees we have another solution!) which means there will be multiple universal maxima with the same values. It's interesting to see if the MIMIC or Genetic Algorithms can perform better or not.

Before we discuss the conclusion of the things found in this study, the hyperparameters used for the tests were:

- Simulated Annealing: starting temperature was 10^1, and cooling was 0.1, ran for 100 iterations.
- Randomized Hill Climbing ran for 100 iterations
- Genetic had a population of 200, 0 mates, and 10 mutations, ran for 100 iterations.
- MIMIC had 200 samples and ran 10 of them, ran for 30 iterations





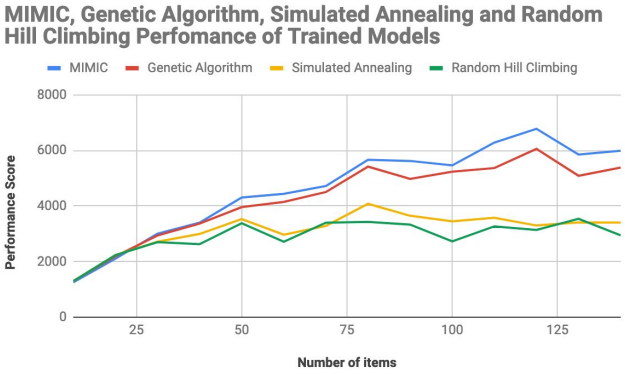
Interestingly enough, since this problem is easy enough, all algorithms performed equally with their performance scores (for the purpose of this test, the higher performance score, the better), but as we see on the left side, Simulated Annealing and Random Hill Climbing had the lowest times for execution, so overall they found the answers the quickest.

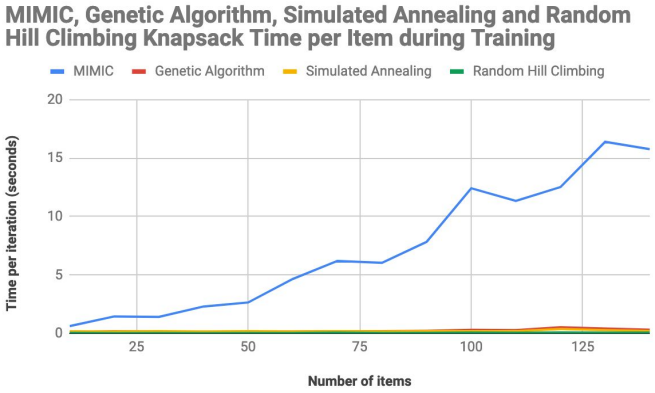
Knapsack

The Knapsack problem is defined as so: given a set of items each having a weight, and given a max weight you are allowed, determine the most items you could bring so that the sum of their weights is less than the max weight. I thought this algorithm would be very interesting to test out because of the educational nature of the Knapsack problem: this problem is discussed at every University's algorithms class, and has a known dynamic programming solution, and it would be interesting to see how Machine Learning could figure out something a human learns and thinks about, and if it can reach its own answer.

Before we discuss the conclusion of the things found in this study, the hyperparameters used for the tests were:

- Simulated Annealing: starting temperature was 10^2, and cooling was 0.95, ran for 200,000 iterations.
- Randomized Hill Climbing ran for 200,000 iterations
- Genetic had a population of 200, 150 mates, and 25 mutations, ran for 1000 iterations.
- MIMIC had 200 samples and ran 100 of them, ran for 1000 iterations





Based on the performance score (for the purpose of this test, the higher the score the better the model performed), its visible that both MIMIC and Genetic Algorithm performed well, with MIMIC performing the best. Again, if training time is a problem, Genetic Algorithm would do better, but since it's not a problem, MIMIC wins this one. One potential reason these two algorithms do the best is because in Knapsack, similarly to Travelling Salesman, you are building off of smaller subproblems: in this example, given we take some amount of object A, how much can we take of the rest. MIMIC and Genetic work best for that.

Conclusion

As part of this assignment, we explored the different kinds of Randomized Optimized algorithms, and tested them on different optimization problems, and tried to optimize problems we have worked on in the past.