James Corwell
Jcorwell3
OMSCS CS 7641
Randomized Optimization – Assignment 2

Introduction

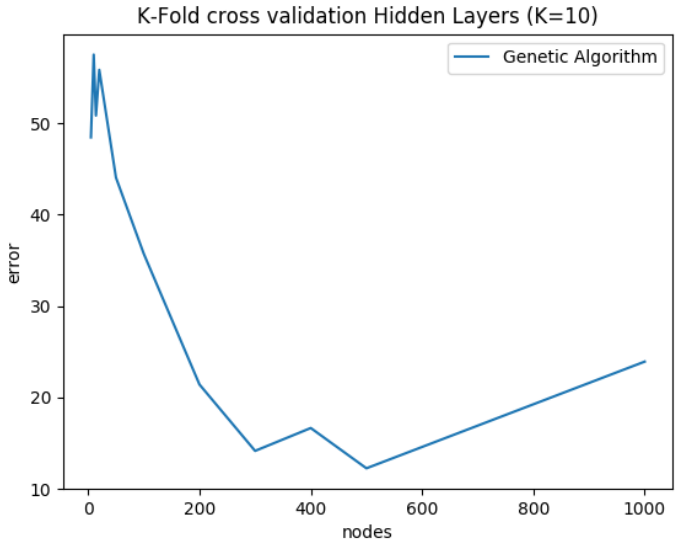
This paper serves two purposes. The first is to try to determine the best weights for a Neural Network using Randomized Optimization techniques as an alternative to Backpropagation. We will also demonstrate the manner in which different randomized optimization algorithms excel when used on different problems. In particular, the usefulness of Simulated Annealing, Genetic Algorithms, and MIMIC, using the Knapsack problem, Continuous Peaks problem, and the Traveling Salesman problem.

Methods

In the generation of all of my data, I used the ABAGAIL collection of algorithms. I implemented some adaptations of ABAGAIL from open source repositories on GitHub. All of the parameters for each algorithm were found by conducting a grid-search of different parameter values.

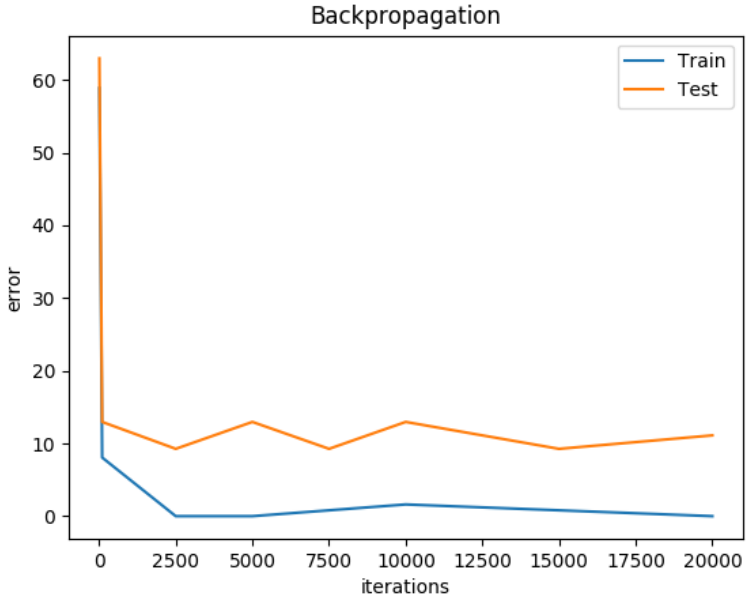
Part I: Tuning a Neural Network using Randomized Optimization Algorithms

The first step for determing the ideal weights of the neural network is to first design the neural network. As my Neural Network from assignment 1 was designed just by “playing” with the parameters, I instead took a more robust approach this time. Using K-Fold cross validation on several different amounts of neurons, we were able to determine the ideal amount of hidden neurons to be 500.



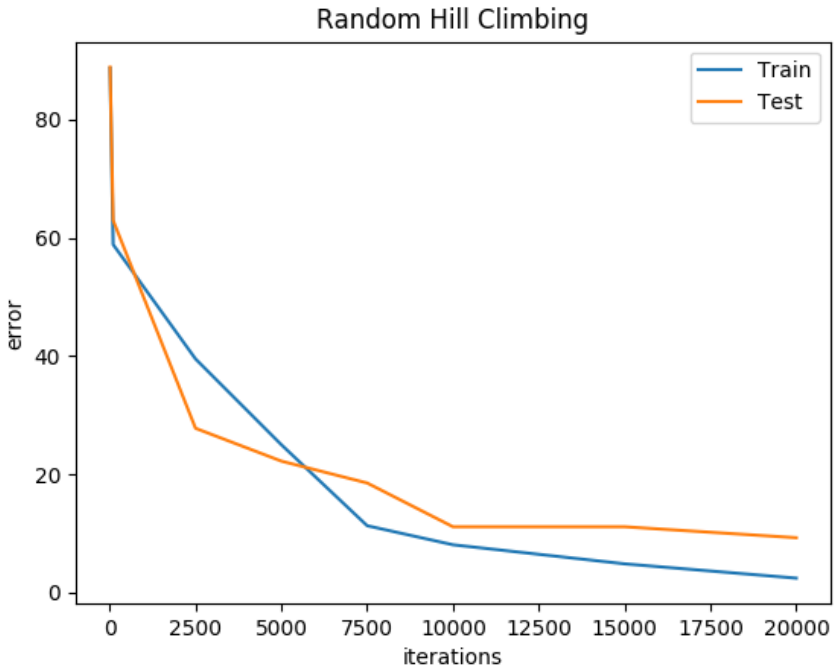
We then defined that the input layer and output layer by the customary method of using the number of attributes, and the number of categories respectively. We set a Max Learning rate = 50, a Minimum Learning rate = .000001, an initial Learning Rate to .1, as these are the fixed values used elsewhere in ABAGAIL.

Backpropagation Revisited



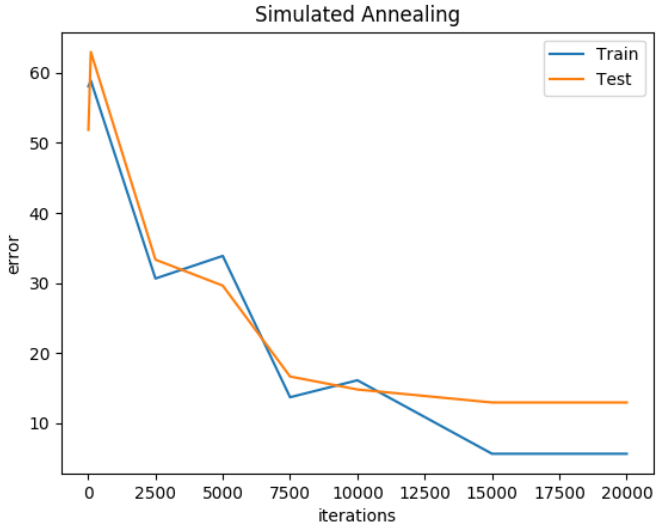
After redesigning my neural network, backpropagation was run again. It converges to near perfect accuracy, giving a learning curve that is highly biased. Also, there is a large gap in the errors of the training and testing scores, showing that backpropagation is also suffering from high variance. However, it did manage to learn the data after only a few iterations.

Random Hill Climbing



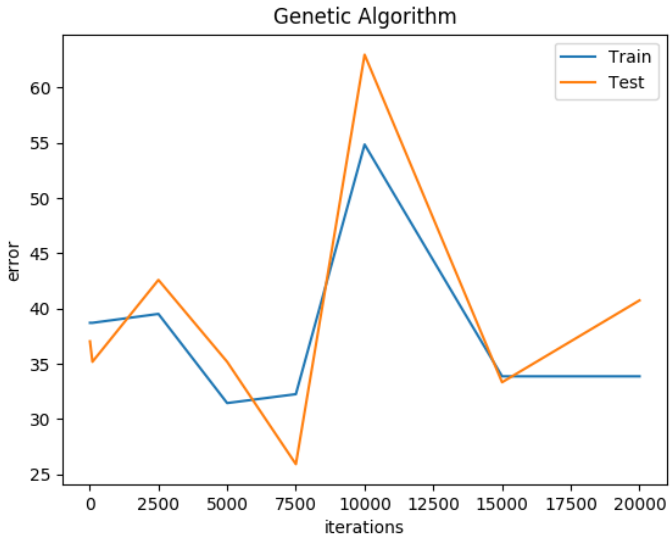
Random Hill climbing performed to a very strong accuracy, rather smoothly and predictably. After 20,000 iterations, it converged to a 2.4% training accuracy, and a 9.3% testing accuracy. The accuracy is high, and the gap between the curves isn't too significant.

Simulated Annealing



For Simulated Annealing, we determined the parameters of the temperature = 10^12, and the best cooling rate = 0.9. Despite Simulated Annealing being a superior algorithm to Random Hill Climbing, we can find that it performed worse at certain higher iterations, which is a bit counterintuitive. After 20000 iterations, it converged to a 5.6% training accuracy and a 12.9% testing accuracy.

Genetic Algorithm

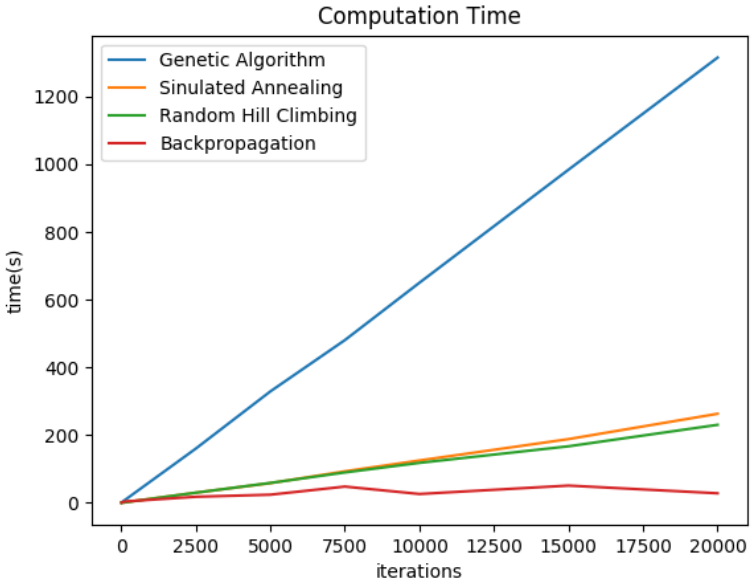


The genetic algorithm was a failure when applied to this dataset. After parameter tuning it was determined using cross-validation, it seems to be far from ideal when used in this manner. We used the parameter population Ratio = .2, Mating Ratio = .02, and Mutating Ratio = .04. The high error that bounces around 33.3%, and this is

a 3-class classification problem—possibly suggesting that this code is trying to classify three classes into two classes.

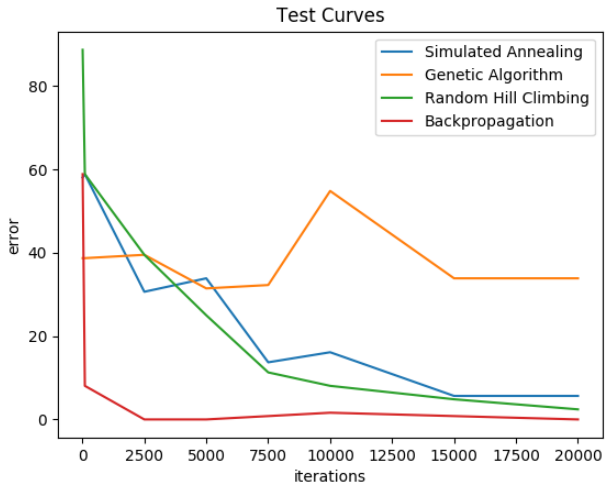
Conclusion

The computation time for Backpropagation was superior. Simulated Annealing was only slightly more expensive than Random Hill Climbing, and by far the Genetic Algorithm was the most computationally expensive algorithm to run – while producing the worst results.



Random Hill Climbing and Simulated Annealing both produced excellent results, and would be the preferred algorithms. If time is short, one might quickly use backpropagation algorithm to demonstrate data trends.

The joint testing curves are as shown:

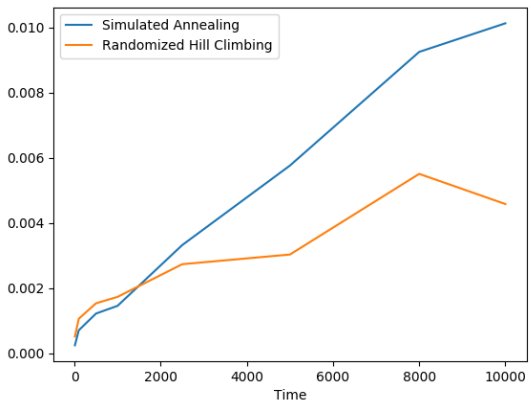


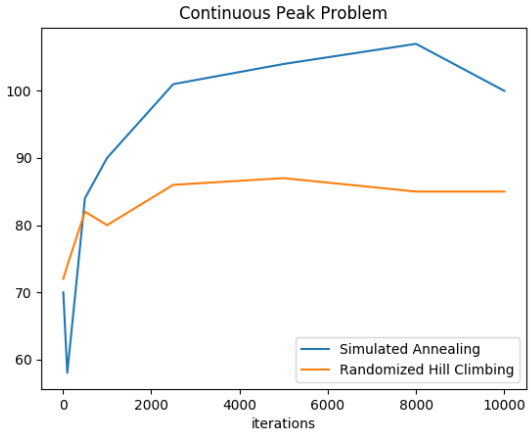
Based on this graph, if your data is large or you have a significant number of neurons, then all algorithms except for backpropagation quickly become unfeasible. If your data set is small or you have a relatively small number of neurons, you can attempt these other Randomized Optimization algorithms at higher iteration ranges.

Part II: Search Techniques Applied to Optimization Problems

Continuous Peaks Problem

Continuous Peaks Problem attempts to find the global maximum based on observation of the local maximum. In this implementation of the Continuous Peaks Problem, we set I set N = 60, and T = 6. Where N is the length of the input vector and T is the bonus limit.



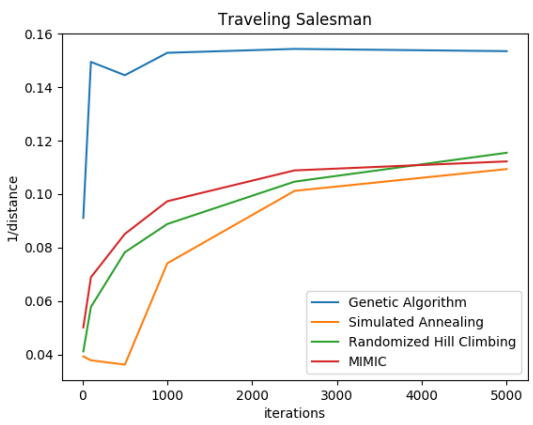


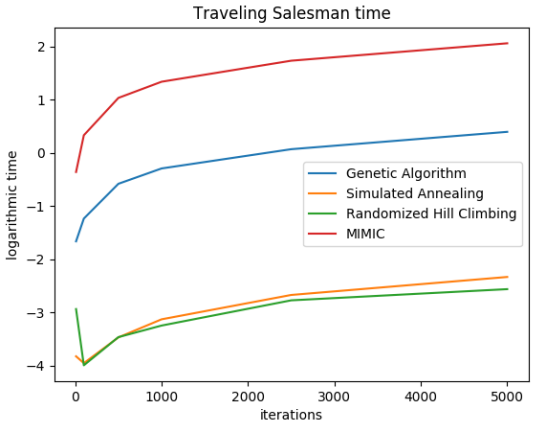
We only compare Simulated Annealing with Randomized Hill climbing in this situation, because the other two algorithms have already been shown to be best. Simulated Annealing

typically doesn't perform as well as Genetic Algorithms or MIMIC. We can see here how simulated annealing is directly better than Randomized Hill Climbing at almost all iterations, but it also takes much more time than Randomized Hill Climbing.

Traveling Salesman

This is perhaps the best-known optimization problem in existence. The question arises if a salesman has to visit a list of cities, in which possible order should he do so. It turns out that to brute force these calculations requires O(n!) complexity.





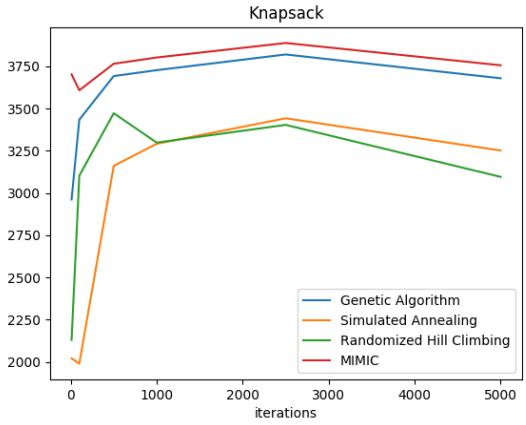
For the traveling salesman problem, we set the value of N = 50. For the TSP, we can clearly see that the Genetic Algorithm performs significantly better than the other algorithms in reaching a peak inverse distance value. In terms of time, it is among the slowest, though MIMIC is still drastically slower.

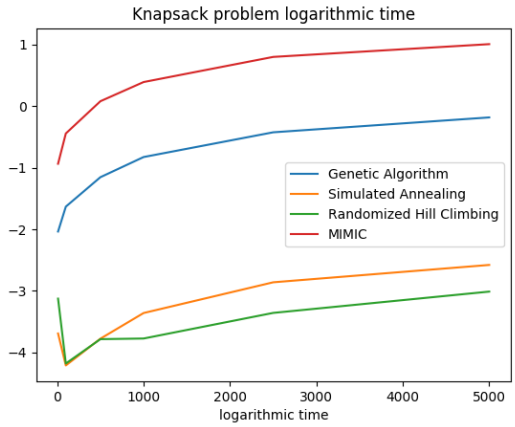
Knapsack Problem

The knapsack problem is a simple and classic optimization problem. Given a bag of fixed volume, and several items each with a given volume and value, how can you fill your back with the most value, to optimize the contents of your knapsack.

When setting up the knapsack problem, we set the number of items to 40, the number of copies of each item to 4, the Maximum weight to 50, and the maximum volume to 50. Knapsack volume is defined as Max_Volume*Num_Items*Copies_Each*.4

These results are as follows:





When looking at the output of the different algorithms tested on the knapsack problem, we can clearly see MIMIC is the top performer on this optimization problem. It also takes a significantly longer computation time than the other algorithms.

When looking at the plot of time, we can see that though MIMIC might give the best results, in some situations it would prove to be wiser to use a Genetic Algorithm just to save on computation time.

Conclusion

This paper has detailed many kinds of Randomized Optimization Algorithms, and showed the idea that there is no “best” algorithm for any purpose. When considering different types of problems, and when computational time is concerned, you should use one of the many possible algorithms that you have available to you.

Overall Genetic Algorithms and MIMIC are the most expensive computationally, while Simulated Annealing and Random Hill climbing tend to be less expensive.

When determining weights of a neural network, you can get some great results using Randomized Optimization, especially with Simulated Annealing and Random Hill Climbing at higher iterations. Though when resources dictate a limited number of resources, the neural-net might need to use backpropagation.