Michael Becker - 903205922

Randomized Optimization

This project consists of two parts; how you can optimize the weights of a neural network with randomized optimization and a comparison of 4 different randomized optimization algorithms and how they perform on three different fitness functions to highlight the differences between them.

For all of the algorithms and fitness function testing, I heavily utilized the ABAGAIL library and visualized the results with Python and Matplotlib.

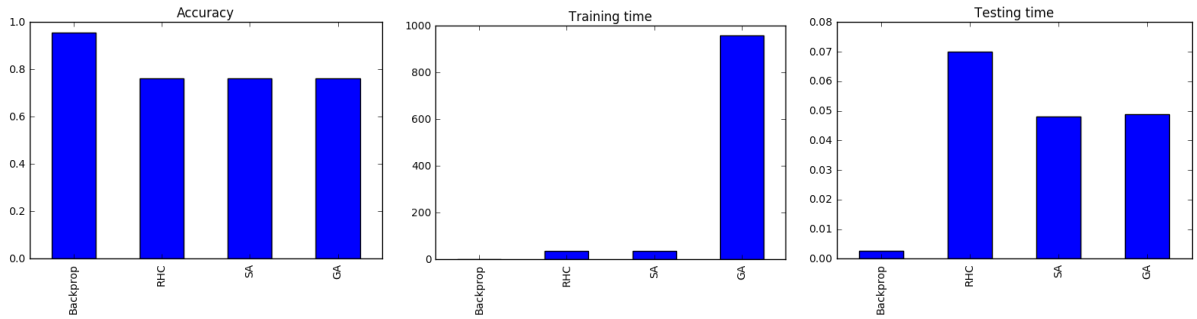
Neural Network Optimization

This section will show how three randomized optimization techniques (randomized hill climbing, simulated annealing, and genetic algorithms) compare against back propagation to determine the optimal weights for a neural network. I will use the dataset and backpropagation neural network results from the first project as a benchmark comparison for the results for each of the randomized optimization algorithms.

Backpropagation

The backpropagation method I used was from the sklearn kit on python on a simple binary classification dataset dealing with employee churn. The results I got from the first project are graphed next to the results for the randomized optimization techniques in the section below along with the analysis.

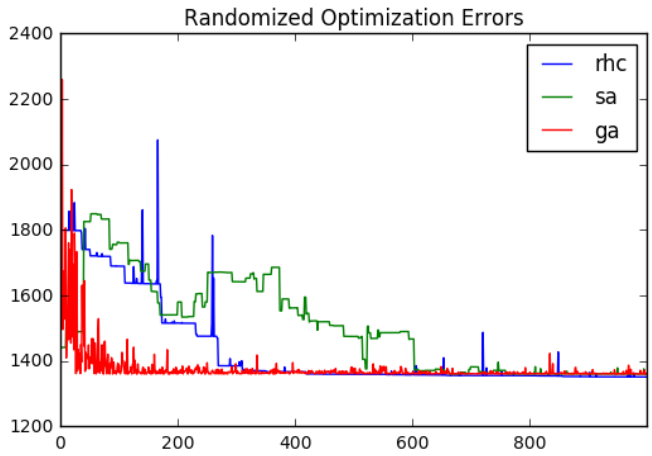
Randomized Optimization Algorithms



The figures above show the overall evaluation metrics between all of the algorithms tested against the employee churn dataset setting the neural network weights. You can see from the charts above that the backpropagation method is clearly the best when it comes to setting the correct weights to reach the optimal accuracy. This is because the neural network used in the sklearn library uses a sigmoid activation function (the fitness function we are trying to optimize) and this is differentiable. This provides an easy way to use the gradient to move towards the optimal solution. Conversely, randomized optimization techniques are most effective when you

are unable to use gradient descent for a fitness function. Moreover, when you have a fitness function that is not differentiable, randomized optimization is the best technique you can use to solve the problem.

Another reason why the backpropagation algorithm provided a more effective accuracy of the dataset is because the concept being learned for this problem is relatively simple so the neural network used was not complex. Neural networks have a tendency to create many local optima as the network becomes more complex; thus leading the backpropagation gradient descent algorithm to get stuck in those local optima. If the problem was more complex and needed a larger network to determine the target concept, I could see that randomized optimization for the weights would be a very useful technique even when the function is differentiable because of the exploitive nature of the randomized optimization algorithms.



The next figure to the left shows the comparison of the error rate between each of the randomized optimization techniques. This visualization helps lend some insight into how each of the algorithms works and provides context into the target concept we are trying to learn for this neural network.

The first observation from this visualization is that the genetic algorithm converges to the optimal error rate the quickest in comparison to simulated annealing and randomized hill climbing. From this observation, we can postulate that the fitness function we are trying to maximize (the most accurate concept for our dataset) has structure that the genetic algorithm can take advantage of. Intuitively this makes sense because the features that we used as inputs to this learning algorithm has relationships to each other allowing the algorithm to use inductive reasoning to generalize a concept. This postulation explains why it takes longer for the simulated annealing and genetic algorithm to take longer to converge because of their greedy nature.

The second observation shows a differentiation between the simulated annealing and the randomized hill climbing algorithms. The simulated annealing algorithm starts along the same path as the randomized hill climbing, but then generates more errors around 300 iteration and starts slowly back down again. This shows an illustrative example of how simulated annealing balances the exploration of the dataset with exploitation that is inherent with randomized hill climbing. As the error starts to increase, the simulated annealing algorithm is moving through crests and troughs in order to find the most optimal solution. In our case, since the fitness function is relatively simple to solve, this is an unnecessary step because the randomized hill climbing algorithm arrived at the global minimum faster, but it is a point worth mentioning. If this

problem had been more complex, the randomized hill climbing algorithm could have been stuck while the simulated annealing would have moved through the local optima into a more global minima.

Conclusion

In conclusion, I want to reiterate two important points about the experiment we ran with neural networks that help to illuminate the advantages and disadvantages to using randomized hill climbing when solving a supervised learning problem. Determining the effectiveness of randomized hill climbing in lieu of the conventional gradient descent algorithm depends on the complexity of the neural network you need to achieve the target concept. While most neural networks are differentiable and can be solved through gradient descent, if the network becomes too complex the chance of getting stuck in a local optima greatly increases. When this happens randomized optimization can help.

Another important point is when using a randomized optimization to set the weights for a given neural network, it depends on the structure of the features you input to the learner to determine which of the randomized optimization techniques to use. If there is structure in the inputs then genetic algorithms will perform best, but more unstructured or less understood datasets will work better with randomized hill climbing or simulated annealing.

Randomized Optimization Comparison

In this section we will compare four different randomized optimization algorithm (randomized hill climbing, simulated annealing, genetic algorithms, and MIMIC) by applying each of them to three different fitness functions that will highlight each of the advantages and disadvantages when using them.

Just like the previous section, I used the ABAGAIL library and the provided fitness functions to perform the analysis across these algorithms. I then used the results to plot them in matplotlib. All of these fitness functions are based off of binary string inputs and are described in detail below.

Four Peaks

The four peaks problem takes in an input vector $\vec{x}$ and runs it through the following fitness function

$f(\vec{x}, T) = max[head(1, \vec{x}), tail(0, \vec{x})] + R(\vec{x}, T)$

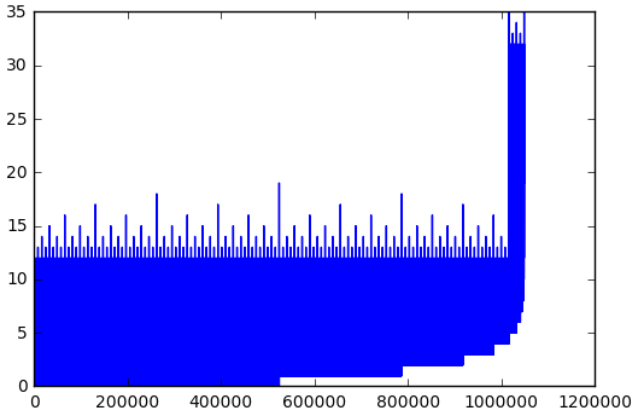
where

$head(1, \vec{x}) =$ number of leading 1's

$tail(0, \vec{x}) =$ number of trailing 0's

$$R(\vec{x}, T) = \begin{cases} N \text{ if } tail(0, \vec{x}) > T \text{ and } head(1, \vec{x}) > T \\ 0 \text{ otherwise} \end{cases}$$

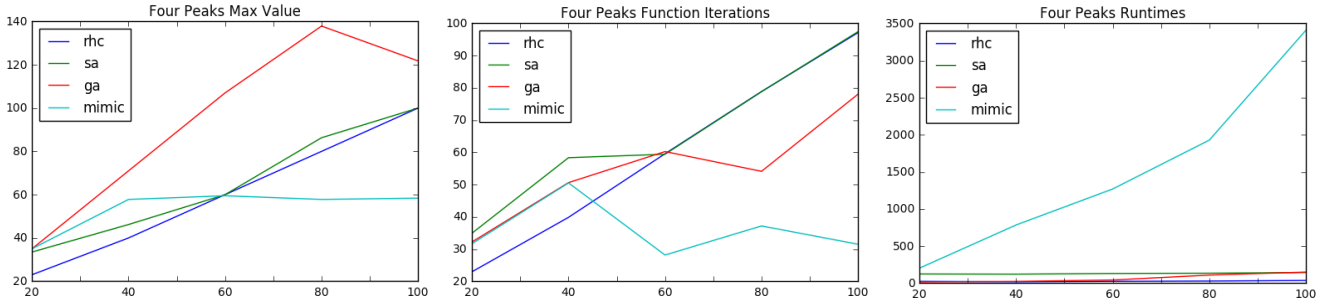
This fitness function will produce 2 global maxima when the number of leading 1's is T+1 and the rest of the string is 0's or if there are T+1 trailing 0's preceded by all 1's. To the right is what the fitness function looks like when it is plotted for an input vector of length twenty and all of the binary variables that can be represented by that vector. This graph clearly shows the two global maxima associated with this fitness function.



Experiment setup

For this experiment, I ran each of the algorithms five times for each N value and took the average since the values can vary on each run because of the stochastic nature of the problem. I used the values N (size of the binary string input vector) = {20, 40, 60, 80, 100} and set T = N / 5 for each of the N values run. I also used a exhaustive grid search to tune the hyperparameters for simulated annealing, genetic algorithm, and MIMIC to ensure that the most accurate max value was reached for each N value.

Results



The figures above show the results between each of the four algorithms for the aforementioned four peaks problem. The first figure shows the max value achieved for each iteration, the second figure shows how many iterations it took to converge to an optimal answer, and the third shows the time it took to run the algorithm.

As you can see from the first figure, the genetic algorithm performs much better than the rest of the algorithms when looking at the what max value was achieved for each of the N values. This is because of the multi-dimensional relationship between the leading 1's and the trailing 0's present in the fitness function. Genetic algorithms can use the relationship between both dimensions to move in the direction of the global maximum and is done specifically through the

crossover function present in the algorithm. Genetic algorithms also do well in this scenario because of the well understood structure of this search space. The GA is able to exploit this search space's structure to better find the global maximum of the problem.

This contrasts the RHC and SA algorithms because each of these algorithms (especially RHC) tend to exploit the data more and rely only on one dimension of the data. This can cause these algorithms to get caught in local maxima often; which happened more often than not for this experiment since this fitness function has a very small basin of attraction. Another reason why SA has a hard time annealing this problem is that the fitness function is relatively flat until the global maxima. As the input space continues to grow, the temperature will start to cool and each move will get caught in a local maxima and will not be able to leave.

Another thing worth mentioning is that the function iterations for GA is relatively low and the runtime is very low. This further proves that genetic algorithms provide a fast, computationally feasible solution for the max value of this particular fitness function.

Continuous Peaks

The continuous peaks problem takes in an input vector $\vec{x}$ and runs it through the following fitness function

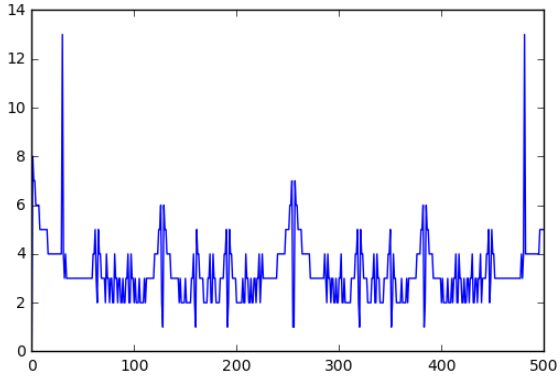
$$R(\vec{x}, T) = max[maxnum(0), maxnum(1)] + R$$

Where

$$maxnum(x, \vec{x}) = \text{ the maximum occurances of character x in } \vec{x}$$

$$R(\vec{x}, T) = \begin{cases} N \text{ if } max(0, \vec{x}) > T \text{ and } max(1, \vec{x}) > T \\ 0 \text{ otherwise} \end{cases}$$

The visualization of this fitness function over a nine character binary string represented from decimal values 0-500 is shown to the right. This chart shows that the problem has 2 global maximums, but has a lot of local optimas within the middle of the input space. One important structure to notice about this fitness function constant flow of hills and valleys that did not exist in the relatively flat fitness function presented in the four peaks problem.

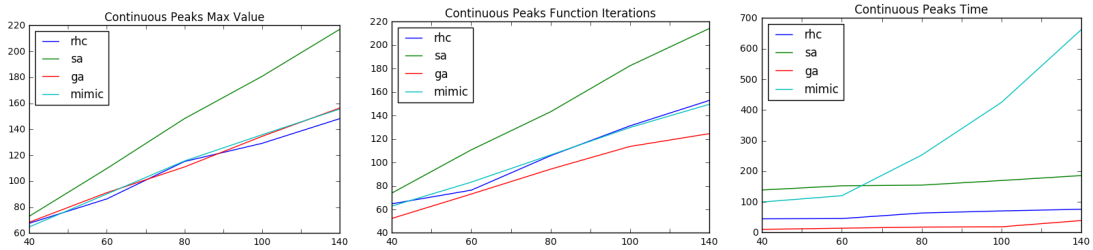


Experiment setup

The setup for this experiment is pretty much the same setup as the four peaks experiment except the N values that I use are { 40, 60, 80, 100, 120}. I still use the same exhausted grid

search over the hyperparameters for each algorithm and take the average over 5 runs for each N. T is always set to N/10 for our experiments.

Analysis



The continuous peaks problem highlights the advantages of using a simulated annealing method over each of the other algorithms. You can see this clearly illustrated by the first figure showing the max value obtained by each of the algorithms over various values of N. The reason that SA is able to solve this problem more effectively than the rest of the algorithms is related to the structure of the fitness function shown in the figure earlier in the section. The key features of the structure I am referring to is the continually oscillating (creating many peaks and valleys) nature of the function and it's relatively unstructured form. SA will thrive in this environment, in comparison to the rest of the algorithms, because of it's ability to both explore and exploit the search space to find the global maximum.

Conversely, since the continuous peaks problem has a relatively unstructured fitness function, the MIMIC and genetic algorithms both have a hard time finding the global maxima. This is because there is no structure for the genetic algorithm to exploit and the knowledge of previous points will not help the MIMIC algorithm.

Another thing to take into consideration is the time and iterations done by each of the algorithms. While simulated annealing did take longer than most of the algorithms to converge on an optimal answer, the time it took was still lower than something like MIMIC. This is something to take into consideration when using SA because this algorithm can theoretically always converge on the right answer, but that convergence can sometimes take an infinite amount of time. That is not the case here since we are looking at relative times and it doesn't seem to take an inordinate amount of time to converge, but it is something to consider nonetheless.

K colors Problem

The k-color problem takes in an input vector $\vec{x}$ and runs it through the following fitness function

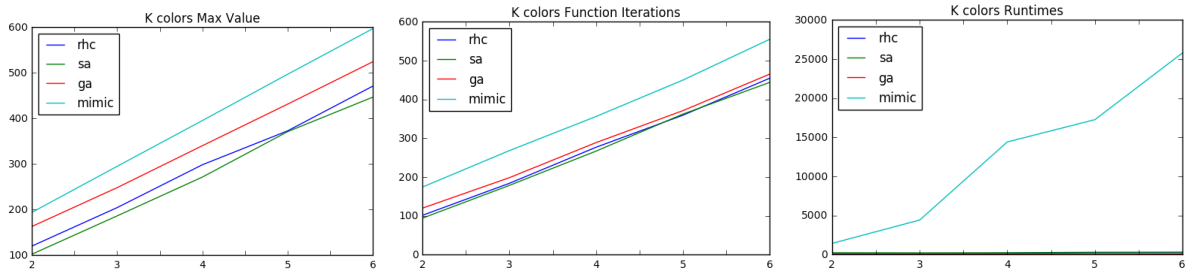
$$f(\vec{x}) = \sum_{1}^{len(x)-1} \begin{cases} 1 & \text{if } x_i \neq x_{i-1} \cap x_i \neq x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

This fitness function is rather simple in that it computes the total number of dissimilar touching bits; illustrating that if you had a graph of N amount of nodes, you can't have k colors touching each other. This fitness function is a great example of a fitness function that relies on the structure of the function to find the optimal solution, rather than being mislead by the individual values at each point in the function. The figure to the right shows the fitness function across all values of binary strings of length 9 where k is 2.

Experiment setup

The setup for this experiment is pretty much the same setup as the previous two experiments except the k color values that I use are { 2, 3, 4, 5, 6 }. I still use the same exhausted grid search over the hyperparameters for each algorithm and take the average over 5 runs for each N.

Analysis



As I mentioned before, the k-colors fitness function greatly highlights the advantages to the MIMIC algorithm because of the structure of the fitness function. MIMIC is able to look at the structure of how the values are distributed and exploit this information to achieve an optimal solution. Rather than looking at individual values coming back from the fitness function and trying to exploit them directly, MIMIC is able to build a distribution about the previous points that it has seen in order to derive a higher level of understanding about the problem. GA is able to do this to an extent, but isn't quite able to achieve the maximum values like MIMC does.

Conversely, RHC and SA are greedy algorithms and all they look at are the individual values coming back from the fitness function in order to exploit the data. This keeps them from fully understanding the structure of the fitness function in order to find the optimal solution like MIMIC and GA.

Conclusion

In conclusion, I have provided three different problems showing that the structure and complexity of search space greatly impacts what kind of randomized optimization technique will perform the best. In situations where there is relatively low structure and often contradictory constraints within the fitness function, algorithms like randomized hill climbing and simulated annealing work the best. Conversely, if the structure of the fitness function is well understood or

finding the global maximum depends on knowing things about previously seen data points, then genetic algorithms and MIMIC tend to converge on the global maximum more often.

Another important aspect to take into consideration is the number of iterations each of the algorithms take for specific search space to converge and the time it takes them to do so. I showed that MIMIC always tends to provide the least amount of iterations in order to successfully converge on an optimal answer, but it also takes the longest amount of time to do so. Though the small iteration count can be used to an advantage when the fitness function is extremely complex to run. This is in direct contrast to the simulated annealing approach where the amount of iterations in order to converge can be extremely high; thus requiring potentially exponentially more iterations making the problem computationally intractable.