Randomized Optimization, Unsupervised Learning, and Dimensionality Reduction

Writeup for Assignment 02 - CS 6741

Magahet Mendiola

ABSTRACT

An empirical analysis of randomized optimization, clustering, and dimensionality reduction algorithms.

1. PART 1: RANDOMIZED OPTIMIZATION

In order to compare and contrast the given randomized optimization algorithms, a classification task was taken from the previous assignment and each algorithm was utilized to train a neural network to perform this task. Implementation of each algorithm as well as the code for running the neural network was taken from the ABAGAIL machine learning library (https://github.com/pushkar/ABAGAIL).

1.1 Neural Network Training

Classification Task.

The dataset used to test our neural network optimization task was the Adult dataset from the UCI repository. This was the second of the two classification datasets used in the supervised learning assignment. To recap, these are samples of personal socioeconomic attributes with a binary classification of gross net income over or below 50k/yr.

This dataset should prove interesting as an optimization task, since we have a fairly clear intuition of the relationship between the attributes and the classifications. From the previous assignment, we could see that attributes such as capital gain or loss had a strong linear relationship to the classification, as did education level. Intuition, or in this context domain knowledge, suggests that the fitness function for our neural network will have a smooth surface with predictable gradients. This is due to belief that having a slightly higher or lower education level would have a slightly higher or lower (mostly predictable) impact on one's net worth. In the case of our neural network, the actual fitness function is the sum of squared errors between the network's output and the actual instance binary values.

The optimization task was run against a training set of 1,000 instances from the adult dataset and final classification error was then measured with a separate testing set of another 1,000 instances. 19 attributes were used from the set and nominal attributes were first converted into separate binary attributes.

Optimization Parameters.

Randomized hill climbing, simulated annealing, a genetic algorithm, and gradient decent were all utilized to train the weights for our neural network. The network consisted of 19 input nodes, 5 hidden nodes, and 1 output node. Each algorithm was given 1,000 training iterations to find an optimum set of weights. Simulated annealing was set with an initial temperature of 1×10^{11} and a cooling exponent of 0.95. The genetic algorithm used a population of 200 with a set of 100 used for crossover and another 10 used for mutation.

Optimization Results.

Using three randomized optimization algorithms, our neural network was trained over 1000 iterations each. The classification accuracy of each of these trained networks is shown in Figure 1. As shown, randomized hill climbing outperformed the rest both in terms of training the network to correctly classify instances and in the time required to train the network. Simulated Annealing was negligibly faster, but the resulting network did not perform as well in the classification task.

This result reinforces our intuition regarding the dataset. Randomized hill climbing is similar to gradient decent in that the algorithm makes small changes in a direction that improves the fitness function result. Since we believe the profile of the adult dataset to be smooth and behaves predictably given changes in the inputs, both gradient decent and randomized hill climbing should move quickly toward a local minimum. In this case, since simulated annealing and the genetic algorithm did not find a solution that performed better than these two, we make the claim that this local minimum is in fact the global minimum.

Training Performance.

It is interesting to note the performance of each algorithm throughout the set of training cycles. This gives a better picture of how each optimization process interacted with the fitness function. In the case of randomized hill climbing (Figure 2) the learning curve appears to smoothly converge toward lower classification error rates. This shows that the algorithm works well in this case in moving the fitness function step by step towards an optimum.

As the error rate does not seem to stabilize over the 1,000 iterations, an additional experiment was performed to see if it could continue to improve the weights if allowed to run for 10,000 iterations. The results are shown in Figure 3. You can see that the algorithm continues to find weights that result in a lower classification error on the training set. However, the resulting network resulted in almost the same classification error when run against the set aside testing set. This result illustrates the previous understanding of neural network performance and it's ability to overfit the training

```
Results for RHC:
Correctly classified 764.0 instances.
Incorrectly classified 236.0 instances.
Percent correctly classified: 76.400%
Training time: 3.709 seconds
Testing time: 0.004 seconds

Results for SA:
Correctly classified 733.0 instances.
Incorrectly classified 267.0 instances.
Percent correctly classified: 73.300%
Training time: 3.023 seconds
Testing time: 0.004 seconds

Results for GA:
Correctly classified 763.0 instances.
Incorrectly classified 237.0 instances.
Percent correctly classified: 76.300%
Training time: 72.068 seconds
Testing time: 0.004 seconds

Results for GD:
Correctly classified 725.0 instances.
Incorrectly classified 175.0 instances.
Percent correctly classified: 72.500%
Training time: 6.560 seconds
Testing time: 0.004 seconds
```

Figure 1: Randomized optimization training on ANN results

s the error rate does not seem to stabilize over the 1,000 iterations, an additional experiment was performed to see if it could continue to improve the weights if allowed to run for 10,000 iterations. The results are shown in Figure 3. You can see that the algorithm continues to find weights that result in a lower classification error on the training set. However, the resulting network resulted in almost the same classification error when run against the set aside testing set. This result illustrates the previous understanding of neural network performance and it's ability to overfit the training data.

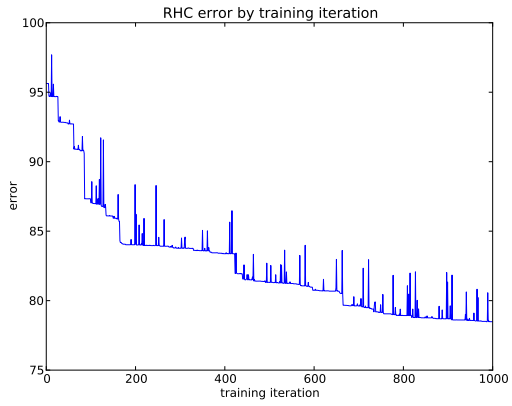

Figure 2: randomized hill climbing ANN classification error by training iterations

data.

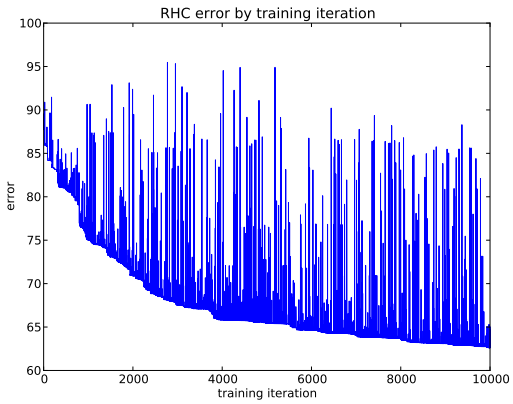

Figure 3: randomized hill climbing ANN classification error by training iterations

Figure 4 shows the results of simulated annealing over the 1,000 training iterations. We see that, unlike randomized hill climbing, the error increases for more than a single iteration, and increases more towards the beginning of the process. When the algorithm fist starts, temperature is still high enough to allow movement toward less optimal results. Around 300 iterations in we see a dramatic improvement in the error. As the temperature cools, the variation in error decreases and the algorithm converges toward local optimum. In this particular optimization problem, the result of this movement in sub-optimal directions most likely caused a delay in reaching an optimal point. However, given a fitness function with a number of local optima, this behavior would have helped to discover remote valleys.

Figure 4: simulated annealing ANN classification error by training iterations

Our genetic algorithm converged very quickly toward an optimum set of weights. However, elapsed time was an order of magnitude greater than any of the other optimization algorithms. This is likely due to the computation involved in maintaining the population and performing crossover and mutation functions. This can be somewhat offset by the fact that the genetic algorithm reached a optimum set of weights

in roughly half the number of iterations, but that still puts elapsed time near five times that of the others.

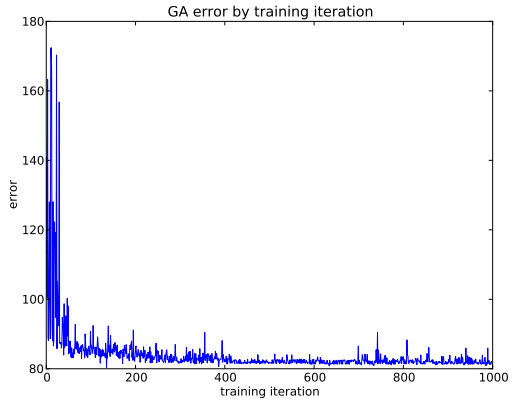

Figure 5: genetic algorithm ANN classification error by training iterations

randomized hill climbing simulated annealing a genetic algorithm MIMIC

1.2 Fitness Functions

In order to examine the properties of each randomized optimization algorithm, a number of experiments were conducted on contrived fitness functions. Each of these functions accepts an arbitrary length bit string and returns a single positive value. In order to visualize the features of these functions, they have each been plotted with the fitness score on the y axis and the input bit string encoded as a integer value on the x axis. Although this only presents a two dimensional projection of the function, it does provide some sense of behavior, including a view of local and global optima.

For each fitness function, a series of experiments were run against each optimization algorithm. First, each algorithm was allowed to train against a fixed input space size of 80 bits. The algorithms were allowed to perform a maximum of 10,000 function evaluations before being terminated. This experiment was repeated 10 times for each algorithm and plotted to allow us to visualize the behavior of each optimization algorithm on the given fitness function as they navigated the output terrain.

The second experiment was setup to evaluate how many fitness function evaluations each algorithm required to find one of the global optima. Each algorithm was allowed to run indefinitely, until it either became stuck at a local optimum or found one of the global optima. In the former case, the algorithm was reset and re-run. This experiment was repeated for differing input space sizes (bit string lengths). The results from this set of experiments was to see how well each algorithm performed at finding the function optimum. It also provided an understanding of how well each algorithm performed given differing input sizes.

1.2.1 Optimization Algorithm Parameters

Each algorithm was given the same set of parameters for each experiment.

Random hill climbing was setup to explore an input change of a single random bit and move in that direction if the fitness score improved. If an improved input was not found in a set number of evaluations (the number of input bits), the algorithm was restart with a new randomized input.

The temperature parameter for our simulated annealing algorithm was set at 1,000. This temperature was cooled by reducing it by 5% each iteration.

Our genetic algorithm maintained a population of 100. 50 of those were used to perform uniform bit crossover. The other 50 were mutated by swapping two random bits in their sequence.

The MIMIC algorithm was set to maintain a population of 100.

1.2.2 Count Ones

The count ones fitness function is well understood and easy to explain. The function simply counts the number of 1s in the input bit string. Although this is an easy heuristic to follow when calculating the function return value, the function's behavior over the input space was interesting to visualize. Figure 6 shows the plot of the count ones function. As we see, there are many local optima and a single global optimum. The function has a upward trend with a stair step like behavior along the way.

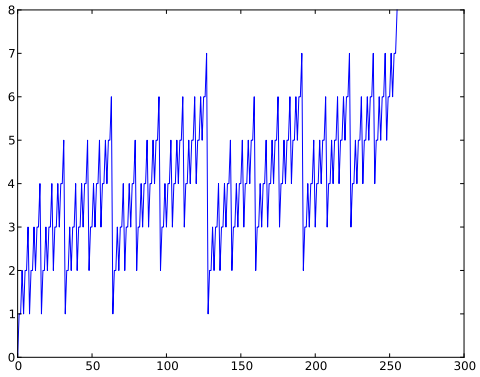

Figure 6: fitness function plot - count-ones

Figure 7 shows how our optimization algorithms performed against the count ones function with an 80 bit input. Random hill climbing appeared to discover peaks quickly. However, the global optimum was not reached in many cases until twice the number of evaluations as simulated annealing. RHC also suffered from it's random restart behavior, which caused many function evaluations at much lower values of the function.

Simulated annealing was clearly the superior algorithm in this experiment. All 10 runs reached the global optimum within roughly 1,000 function evaluations, which represents $1/1{\times}10^{21}$ of the total input space. The genetic algorithm and MIMIC behaved well, making progress with each evaluation. However, they both required an order of magnitude greater evaluations to move toward the global optimum.

Given the relatively smooth trend of the function toward a global optimum, simulated annealing seems to have been able to explore the space well enough to move in that direction. As we will see later, finding a suitable temperature parameter is crucial for allowing the algorithm to move out of local optima. However, high temperatures keep the opti-

mization task geared towards exploration longer than is necessary. In the case of this experiment, we happen to stumble on a good balance between exploration and exploitation.

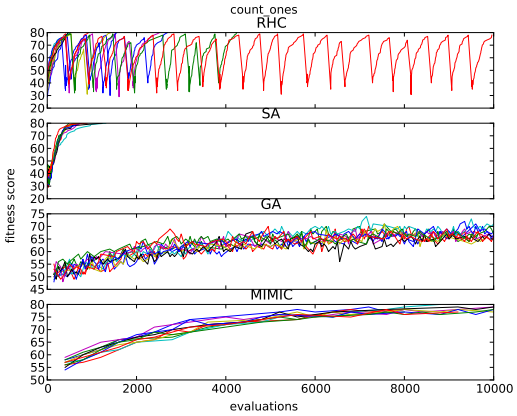

Figure 7: fitness score by iteration - count-ones

Each algorithm performed reasonably well at finding the global optimum for the count ones function at various input sizes, with the exception of our genetic algorithm. Given the somewhat unstructured and independent nature of the input bits, this is not surprising. MIMIC seemed to do well in this regard, despite the lack of structure. We could understand this to be a fortunate side effect of the probabilistic sampling, which causes the algorithm to perform similar to simulated annealing. Figure 8 shows this behavior.

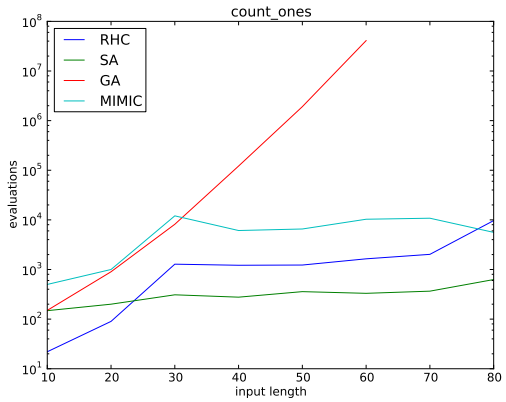

Figure 8: iterations to find max - count-ones

1.2.3 Four Peaks

The four peaks evaluation function is essentially a count of the leading 1s and trailing 0s in a bit string, with a bonus for having at least an arbitrary number of each. The plot of this function shows there are two global optima and a large number of local optima. There is also a point along the plot where values shift upwards. This is the point when the leading 1s reach the minimum number to get the bonus.

Our genetic algorithm was able to find reasonably high valued input strings with a small number of function evaluations (Figure 10. This makes sense given the structure of the

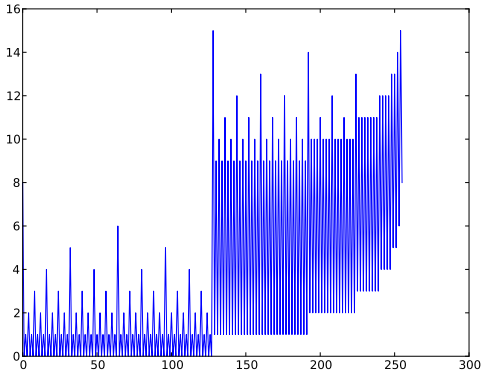

Figure 9: fitness function plot - four-peaks

input space and the fact that the algorithm would quickly evolve the population to include bit strings with better features (longer sets of leading 1s and trailing 0s). However, this algorithm did seem to struggle to find one of the two global optima.

This function also highlights the shortcomings of simulated annealing. Our plot shows that 7 out of 10 runs became permanently stuck in a local optimum. This is likely due to the sharp features of the landscape and insufficiently low temperature and overly rapid cooling rate. This caused the algorithm to get caught on a lower range of peaks.

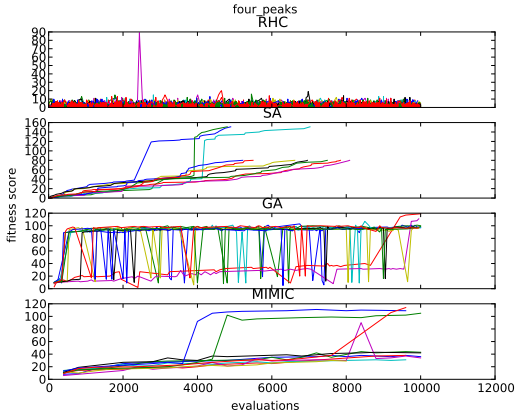

Figure 10: fitness score by iteration - four-peaks

Figure11 shows that although our genetic algorithm was able to quickly find input strings with high values, it required many more iterations to find the global optimum for a given input size. Conversely, despite the impressive figures for simulated annealing, we noted in the previous experiment how that algorithm was too likely to get caught in a local optimum. It required many more iterations of this experiment to get simulated annealing to find the global optimum at all.

MIMIC seemed to behave well in this case. Although it did not locate optimal values as quickly as the other algorithms, it showed steadily improving results over time and did not have the same difficulty with local optima. Also, in

terms of iterations to finding the global optimum, MIMIC performed significantly better than the genetic algorithm for higher input sizes.

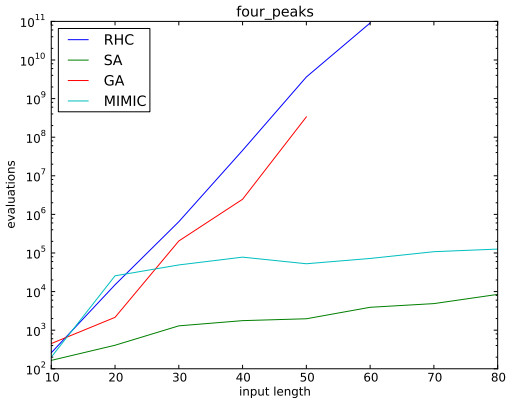

Figure 11: iterations to find max - four-peaks

1.2.4 Flip Flop

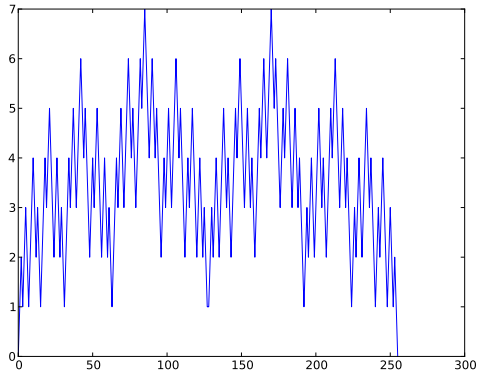

Figure 12: fitness function plot - flip-flop

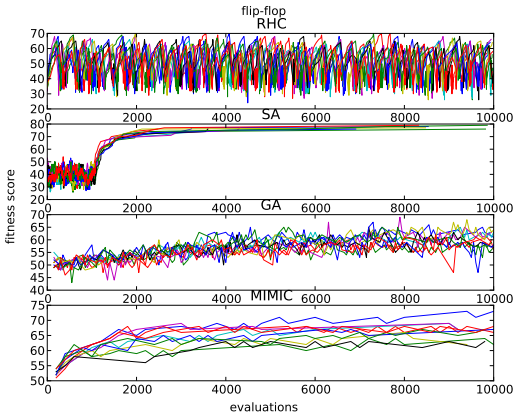

Figure 13: fitness score by iteration - flip-flop

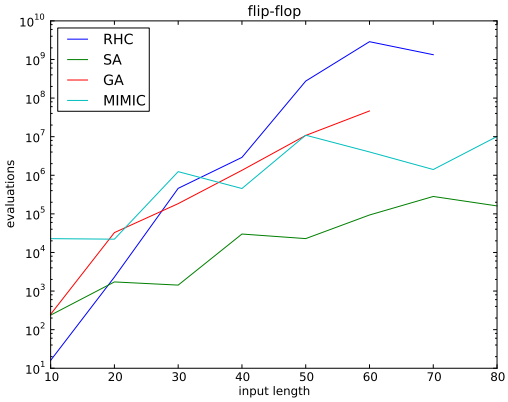

Figure 14: iterations to find max - flip-flop

2. PART 2: UNSUPERVISED LEARNING ALGORITHMS

2.1 Clustering

k-means clustering Expectation Maximization

2.2 Dimensionality Reduction

PCA ICA Randomized Projections Any other feature selection algorithm you desire

3. REFERENCES

[1] K. Bache and M. Lichman UCI Machine Learning Repository 2013 http://archive.ics.uci.edu/ml University of California, Irvine, School of Information and Computer Sciences