Randomized Optimization

Yi Zhao

[Yzhao644@gatech.edu](mailto:Yzhao644@gatech.edu)

Introduction

Optimization is a technique to find the best solution with certain conditions or limitations. It is researched in a lot of area from economy, mathematics to computer science. In computer science area, there many algorithms developed to solve certain problems. For example, dynamic programing algorithm to solve Knapsack problems. In this report, I focused on research on four algorithms. randomized hill climbing, simulated annealing, genetic algorithm and mutual-information-maximizing-input-clustering (MIMIC). I used the above algorithms to solve four optimization problems. They are 4 peak, N-Queens, Knapsack problems and a neural network to solve adult dataset.

The first three problems are designed to show the advantage of individual algorithm and to explain and compare the hyper-parameters in details for the later research for neutral network. Which is again trying to find the best parameters for optimization algorithms and compare with gradient boosting problem.

I started with description on how the algorithms work and how some hyper-parameters are going to affect the algorithm. Followed is my initial thought on the performance of the individual algorithm on each problem. I then showed my process on deciding the hyper-parameters and comparison between different algorithms on fitness, computational time, and complexity. The second part of the assignment is to implement three algorithms on neutral network to solve the adult data.

The library I use is mlrose-hiive.

Keyword: optimization, MIMIC, Knapsack Problems

1. Algorithms

1.1 Random Hill Climbing

Random Hill Climbing comes from Hill Climbing technique, which is method that excels in find the local optima. It starts with a solution and trying to move around to find a better neighbor to improve the solution. The search stops when either maximal iteration or attempts reaches. The highest value of this search is taken as global optimal.

The random come when the algorithm is restarted with another random position to find a new local optimum. The result is the largest of all local optimums. The hyper-parameter to train is the number of restarts of the algorithm.

1.2 Simulated Annealing

Simulated Annealing is algorithm that use ‘explore and exploit’ mechanism to find the global optimum.

The analogy to annealing process is the algorithm starts with a higher threshold (initial temperature) to explore. The threshold is decrease as the temperature goes down, all the way to minimum temperature, according to a decay method (Exponential or Geometrical).

For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to exact algorithms such as gradient descent, Branch and Bound.

There are two hyper-parameters to tune for SA, init\_temp and decay. Init\_temp decides how widely do you want to explore, and decay determine how fast you want to decay to only search for local optimum.

1.3 Genetic Algorithm

Genetic Algorithm technique uses evolutionary algorithms to generate new kids based on previous parents’ generation. The advantage is information are transferred from parents to next generation, through techniques like mutation, cross over and selection.

There are multiple hyper-parameters to tune for GA. This report focuses on two of them. Population size, which decides how many parents are selected to generate newer generation. And mutation rate, which decides how to get new solutions which also has some feature from previous generations.

1.4 Mutual-Information-Maximizing Input Clustering (MIMIC)

This is another technique in Randomized Optimization field. The idea is to generate the description of certain parameters and only take the better half (or some other portion, Keep percentile) for next optimization. It gets input parameter’s distribution from second order statistics of the input data, which is computationally simple. It also has a memory mechanism to transfer learnings from previous iterations to next ones.

The hyper-parameters I focused on are Population size and keep percentile. They are used to decide how much information are gathered to estimate parameters’ distribution and how much information are kept for future iterations.

2. part 1: problems and process of training

In this part, I will introduce the problems I choose to show difference between algorithms and hyper-parameters. How the hyper-parameters are chosen with individual algorithm. An overall comparison of individual algorithm with best hyper-parameter combination in terms calculation time, fitness, and complexity of the problems.

The grid search method I used is the runners within mlrose\_hiive.

I also modified the source code to keep track of running times per iteration. And used customized fitness function to keep track of fitness per evaluation.

2.1 N-Queens Problem

N-Queens Problem is to put N numbers of queens on the chess board so that they cannot attack each other. It’s is a discrete optimization problems. It has multiple optimums, but the maximum fitness function is unique. For a 8\*8 chess board, it should be 2C8/2, which is 28.

The uniqueness of the problems is that all the optimums are reached at the same level (28) and the basins between the optimum are not very wide. In another word, it’s easy to change from one solution to another better and very hard for algorithms to stuck on local optimum.

2.1.1 Solve N-queens with Random Hill Climbing

As mentioned earlier, one hyper-parameter to tune here, namely the restarts time. One thing to notice is that even with no restarts, the algorithm could succeed in finding the optimum. This is same because the N queens problem’s local optima are located close to each other and many of them are global optimum.

|  |  |  |
| --- | --- | --- |
| **Restarts** | **Sum of Time** | **Average of Fitness** |
| 0 | 1.1968895 | 25.66666667 |
| 10 | 20.4609738 | 24.42424242 |
| 25 | 250.0702527 | 24.51282051 |

Since it’s required to use the random hill climbing algorithm, I am using restart = 25, since it’s giving the higher average of fitness. And the reason for that is since there are multiple optimum, having it restart multiple time making the algorithm converging faster and making the average fitness larger than 10 restarts.

2.1.2 Solve N-queens with Simulated Annealing

4 peak is designed for ga. <https://www.ri.cmu.edu/pub_files/pub2/baluja_shumeet_1995_1/baluja_shumeet_1995_1.pdf>

No free lunch, mimic needs some specific randomness for better converge.

Basic construction

Use customized function to keep track of evaluation.

Modify source code to keep track of runtime.

And iteration.

Random seeds

Mutation rate in neural network

3. part 2: neural network

optimization is generative learning and gradient boosting is more like descriptive learning.

Optimal could be a lot, hard to be generative with the data. Greedy algorithms suffers more than explore and exploit and more than distribution method.

Reference

<https://en.wikipedia.org/wiki/Hill_climbing>

<https://www.ri.cmu.edu/pub_files/pub2/baluja_shumeet_1995_1/baluja_shumeet_1995_1.pdf>

<https://www.cc.gatech.edu/~isbell/papers/isbell-mimic-nips-1997.pdf>

https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf

Plots

1. fitness vs iteration

2. fitness vs eval

3. compute time vs iteration

4. compute time/iteration time vs complexity/input size?

5. queens

Four peaks

Find ones?

Knappack

evaluate\_population\_fitness

set\_state

best\_neighbor

optimization is generative learning and gradient boosting is more like descriptive learning.

Optimal could be a lot, hard to be generative with the data. Greedy algorithms suffers more than explore and exploit and more than distribution method.