Randomized Optimization – Machine Learning CS7641

Tichakunda Mangono, GTID: tmangono3

March 3, 2019

Abstract

This paper will apply four randomized search algorithms - Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithms (GA), and Mutual-Information-Maximizing Input Clustering (MIMIC) - to the same problem i.e. comparatively analyze the performance and computational complexity of each algorithm as the solution to finding the optimal weights on a feedforward Neural Network over a familiar data. To develop further intuition into the advantages and disadvantages of each randomized search algorithm, this paper will propose 3 well-known problem domains each highlighting the pros and cons of one randomized optimization method over the others.

This analysis relies heavily on the Python library, mlrose for optimization as well as the standard python machine learning stack – SciKitLearn, Keras, XGBoost, Pandas, Numpy, Matplotlib, and Seaborn.

Introduction

In general, optimization is finding the value in the input space which corresponds to the maximum or minimum value in the output space based on some objective/fitness function which maps instances from the input space to the (real-valued) output space. While there are several **non-randomized optimization methods**, they only work under specific assumptions. For example, **Enumeration/Generate & Test** for complex functions over small input spaces; **Calculus** for differentiable, smooth, and continuous functions; and **Analytical** methods e.g. Newton Raphson for single-optimum, iteratively improving, and differentiable functions. Therefore, when some or all of the conditions above are not met i.e. when the input space is large, or the derivative does not exist or is hard to find, or the fitness function (or objective) is complex with possible multiple optima, then we need to use **randomized optimization methods**. Randomized optimization techniques rely on randomization in the steps to find the "best" solution to a given optimization problem. These method include Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithms (GA), and Mutual-Information-Maximizing Input Clustering (MIMIC). For this paper, "best" refers to the maximum value of the objective function.

Part I: Comparative Analysis of the Performance of Randomized Search Algorithms in Finding the Weights of a Neural Network

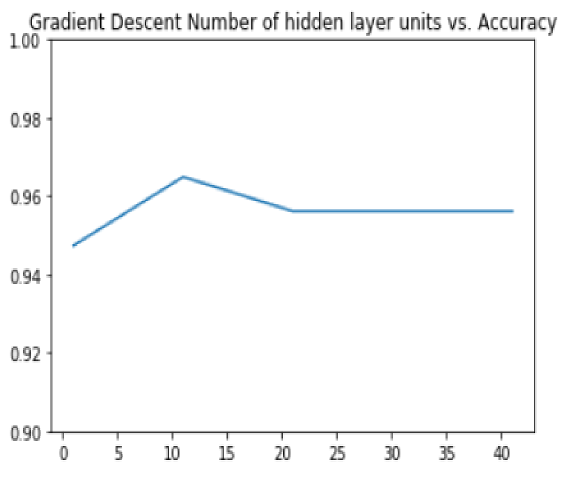
The Data

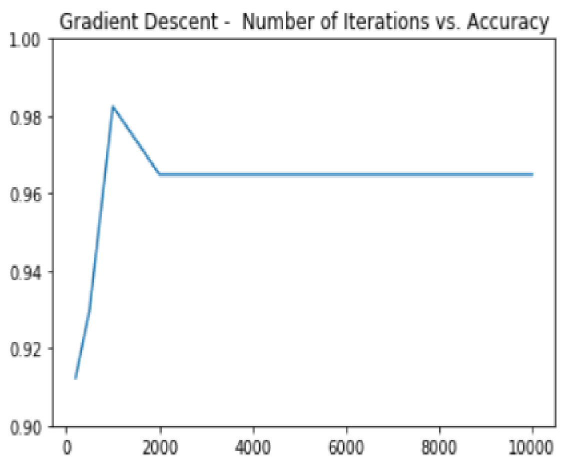
The dataset selected for this analysis the classification problem: **breast cancer diagnosis** (referred to as “*diagnosis*” for the rest of this report) – a labelled dataset of 569 instances with 30 features computed from a digitized image of a fine needle aspirate of a breast mass, describing the characteristics of cell nuclei present in the image. The diagnosis dataset is interesting as it represents the major topic predictive disease diagnosis. It is a set of carefully computed secondary features of the area of interest (cell nucleus) in an image so it has a real-valued, continuous, and heterogeneous input-space.

For the methodology, the data is split into 80% for training and 20% for testing, with 5-fold cross-validation training. First, each randomized algorithm is trained on a neural network with two hidden layers and a Rectified Linear unit as the output layer. The number of iterations and perceptron units is then varied to investigate how performance and computational complexity changes with these hyperparameters.

Gradient Descent – For objective comparison

The weight optimization method of gradient descent was used to optimize a neural network over the data. To find the best configuration for optimization under this algorithm, the number of hidden units in the single hidden layer were varied from 1 to 50 (with 10-unit increments), while the number of iterations were varied from 1 to 10000 (with irregular increments). The rectified linear unit (relu) was used as the output layer with test/validation accuracy as the measure of performance. Independently the configuration of 10 units in the hidden layer resulted in >96% accuracy while using 1000 iterations got over 98% accuracy in this breast cancer diagnosis problem. Combined, a neural network with 10 units in hidden layer and 1000 iterations achieved 98% accuracy. This was the best performance for the Neural Network using the gradient descent method. Below the charts show the independent tests run to find the best number of perceptrons in the hidden layer and the best number of iterations. This now gives us a good point of comparison for the randomized algorithms.



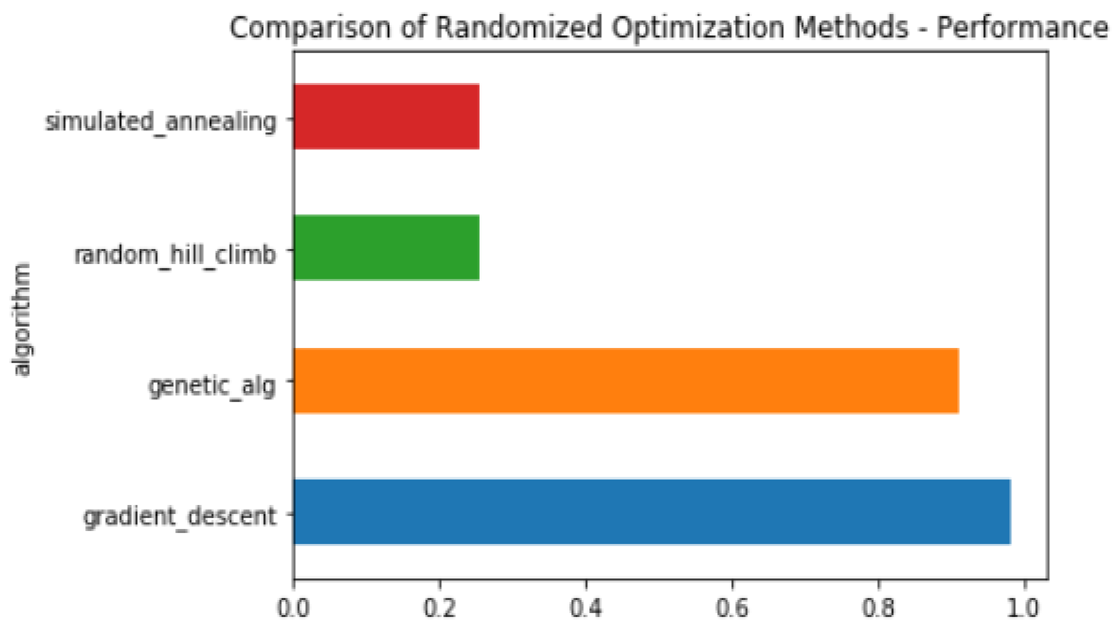


Comparing All the Randomized Optimization on Time and Performance:

After finding the best gradient descent performance hyperparameters, the same hyperparameters were used on the three of the randomized optimization algorithms in order to compare performance as well as computational complexity (or time to convergence). This would allow for an initial comparison before varying and improving any of the randomized optimization algorithms. So, keeping the number of units in the hidden layer to 10, and the number of iterations at 1000 with default values for all the other hyperparameters in mlrose package, the following observations were made.

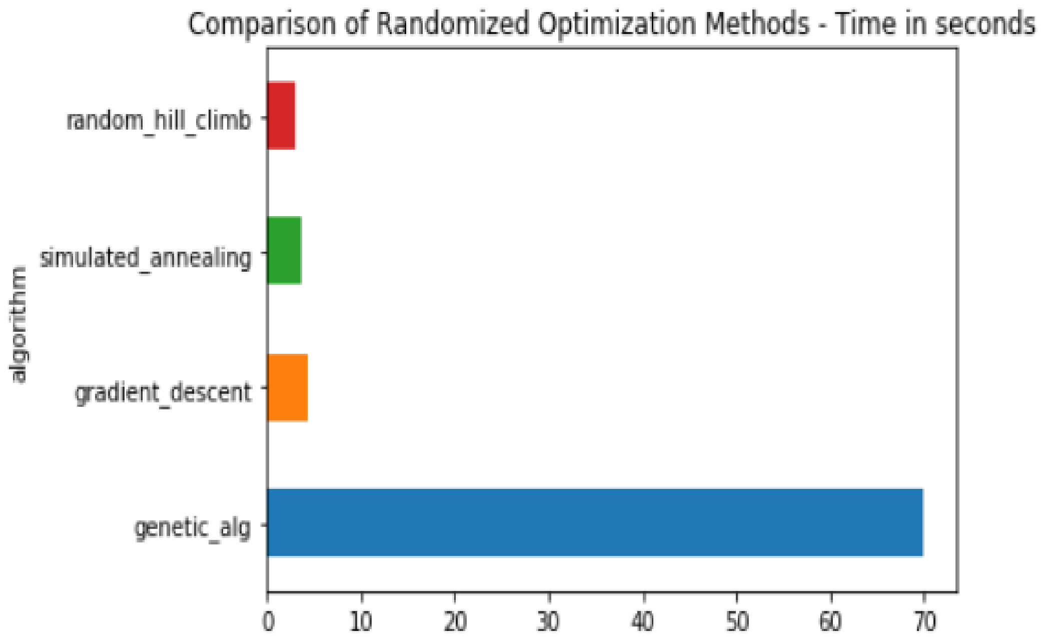
Performance Comparison – Randomized Optimization vs. Gradient Descent on Neural Network

Using classification accuracy as a comparison metric for the performance without changing the initial hyperparameters at which the gradient descent neural network was optimized, all the randomized optimization methods performed worse than gradient descent. While Random Hill Climbing and Simulated Annealing performed significantly worse than the 98% gradient descent benchmark (both achieving 25%), the Genetic Algorithm performed very well against the gradient descent algorithm, achieving slightly above 91% accuracy.



Computational Time Comparison – Randomized Optimization vs Gradient Descent on Neural Network

Using time (number of seconds) to converge to a solution as the measure of comparison, it is very clear that while the genetic algorithm perfoms close to the gradient descent (and significantly above simulated annealing and randomized hill climbing) on accuracy, its computational time (70 seconds) is between 16 and 24 times worse (longer) than any of the other algorithms. So while it achieves good accuracy, the genetic algorithm sacrifices a lot of computational power to converge to a good solution. On the other hand, both simulated annealing and randomized hill climbing are slightly faster than gradient descent but within the same order of magnitude.



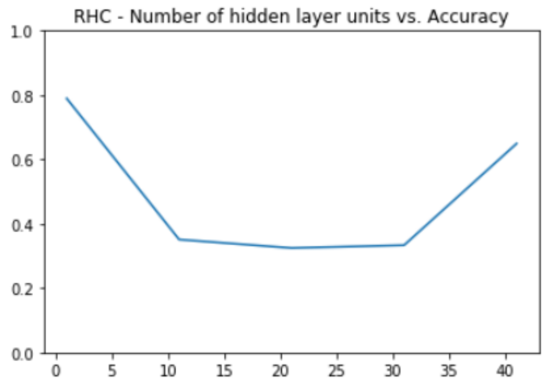
The table below shows the exact numbers from the two charts above:

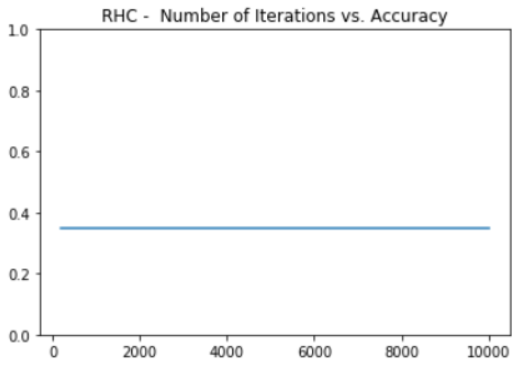
	accuracy	time_seconds
algorithm		
simulated_annealing	25.4%	3.8 s
random_hill_climb	25.4%	2.9 s
genetic_alg	91.2%	69.9 s
gradient_descent	98.2%	4.3 s

So by keeping the hyperparameters fixed we have found that randomized hill climbing and simulated annealing are fast but result significantly lower accuracy than gradient descent while genetic algorithms have high accuracy, closer to gradient descent but take a prohibitively long time to converge. For further investigation, we could now let the hyperparameters change for each algorithm to improve on either their performance and/or time.

Randomized Hill Climbing (RHC)

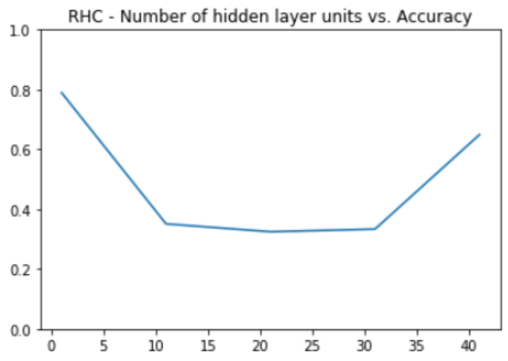
Is an iteratively improving algorithm that takes the step which increases the fitness value by the highest (choosing from among the neighbors of a randomly selected instance). As such, RHC is an instance-based method. Once an optima is reaching, RHC restarts again to ensure that this optima is not local, but that it is indeed global. The test accuracy for RH starts high, decreases and then increases as the number of units in the hidden layer increases. The accuracy does not change with number of iterations, since RHC does not remember information and is based on instance learning.

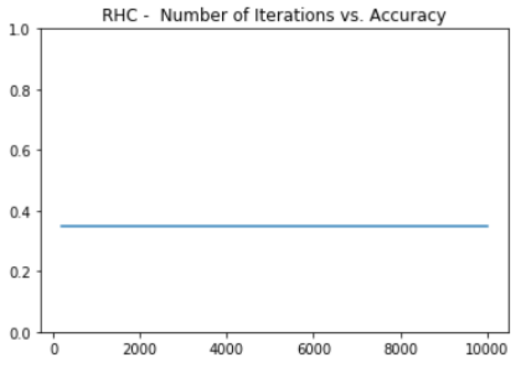




Simulated Annealing (SA)

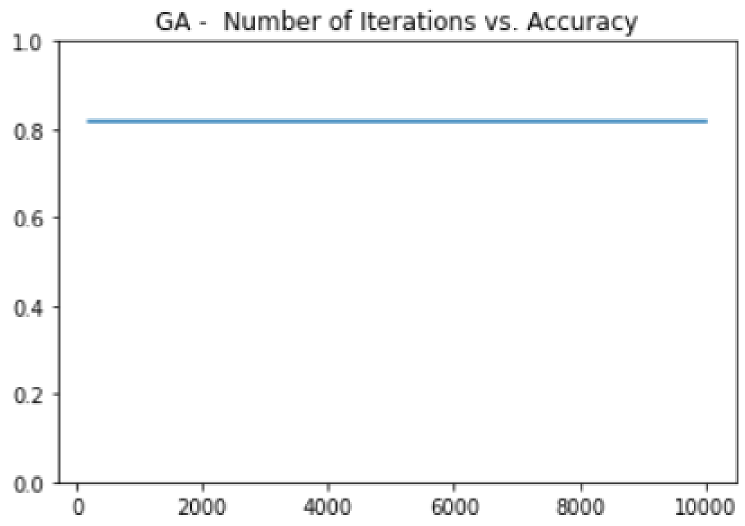
SA is very similar to RHC, except it adds an extra step which allows the algorithm to jump to a different neighborhood (with a certain level of probability) even if it doesn't improve the fitness function immediately. As a result, this leads to an algorithm which both expoits and explores the input space to come to convergence. Like RHC, SA also ch.anges with the number of units in the hidden layer but is indifferent to the number of iterations





Genetic Algorithms (GA)

GA models the biological processes, such as evolution. It starts from an initial population, p1 and computes the fitness function over all instances in p1 and then systematically grows the population with parents producing offspring through a process of crossover (combining aspects of the two parents) and mutation (small random changes which happen with a certain probability). GA accuracy is indifferent to the number of iterations but changes when the population size and the mutation probability are tweaked.



Mutual-Information-Maximizing Input Clustering (MIMIC)

MIMIC is good for directly modeling the distribution of instances, conveying the structure and potentially refining the model. This model was not tested against the gradient descent since it was not required for this paper/assignment.

Part II: Optimization Problems Domains and the relative advantages and disadvantages of all 4 randomized search algorithms

In this section, all 4 randomized optimization algorithms are applied to 3 common problem domains to highlight their strengths and weaknesses. For this purpose, the 3 problems chosen are the i) Travelling Salesperson, ii) 4-peaks problem, and iii) Knapsack Problem

i) Travelling Salesperson

This problem imagines that a salesperson wants to sell goods to a given number of locations, say n, such that they will start and end at the same place and only visit every other location once, thus achieving the shortest trip. This is an NP-hard problem because as n, the number of locations increases it becomes impossible to evaluate the solution within a reasonable amount of time. As such, we cannot not manually or analytically solve it and it is a good area to apply the randomized optimization methods.

The Genetic Algorithm will do well in this problem as it can learn the structure of the problem well.

ii) 4-Peaks Problem

This is a problem similar to the one provided in the Udacity Machine Learning lectures where there are 4 peaks over a discrete set of inputs from 1 to 28 with the 4 peaks falling unevenly across this set of instances. One condition is that one of the peaks must be higher than the rest so that it can be identified as the global optimum. The problem is to use randomized optimization to find this global optimum, effectively making this problem one of maximizing.

This problem is interesting because it may have several applications in the real world particularly when analyzing the surfaces of materials, geography of particular areas with the goal to direct how to traverse that area e.g. for air travel.

For this problem, SA will have the advantage that it will not get stuck any one of the local optima without finding the global optima. SA has a higher chance of getting unstuck since it will jump out of the sticking points to investigate other instances (exploration vs. exploitation) thus increasing the chances of finding the global optimum. As such, convergence to the maxima depends on the size of the attraction basin, and for SA it also depend on the probability of exploration (the higher this is, the more likely SA will avoid getting stuck). This will allow it to detect and jump in-between the 4 optima.

As for GA and MIMIC, these are both distribution-based methods and will converge to the right optima. However, they will take a long time to converge given that they have to form a basis for the distribution when searching for the maxima.

iii) Knapsack Problem

This problem optimizes the total value of the distinct items selected (sampled with replacement) and placed in a knapsack without exceeding the weight limit imposed on the knapsack. This problem is very practical as it has applications in everyday uses where decision-making needs to be optimized like literally filling up a knapsack, least wasteful ways to cut materials, and creating the most valuable, least risky portfolio for investment.

In the general case, the knapsack problem is NP-complete as there is no algorithm that will converge and still be reasonably fast every time, especially when you can place multiple instances of the same item. However, if required to only place at most one unit of each item, then while trivial the problem then becomes solvable in polynomial (or less) time.

Given the problem above, I would expect MIMIC to perform very well in this challenge because regardless of the limit, weights and number of items, it will learn the specifics of the problem and use the learned distribution structure to find the optima. Genetic algorithm will also perform very well given that it also relies on the distribution of the instances.

RHC and SA, being instance-based algorithms will likely perform worse than MIMIC and GA. This is because there are many different configurations and combinations of the items that need to be explored and each configuration theoretically has its own peak, thus increasing the probability that RHC and SA may find suboptimal solutions.

However, as expected MIMC and GA will require more computation time than RHC and SA.

Summary

When comparing randomized search algorithms to gradient descent on a neural network with a binary classification dataset, gradient descent was found to product the best accuracy in a reasonable amount of time while randomized search algorithms either took prohibitively long to converge (genetic algorithms) or resulted in low accuracy (randomized hill climbing and simulated annealing). So in general, gradient descent is the best-in-class method for finding the weights of neural network.

However, when the hyperparameters are customized to each randomized search algorithm, the performance of individual algorithms when compared to gradient descent fluctuates and can be improved.

Given the problems with different structures, all four randomized search methods: RHC, SA, GA, and MIMIC will have different advantages and disadvantages. While RHC and SA are good algorithms, they may get stuck or even increase computational complexity when there are multiple optimas and when the size of the attraction basins varies. On the other hand GA and MIMIC are good for problems where the algorithm can learn to narrow down the search field based on the previous steps, therefore increasing the likelihood of optimal convergence.

Ultimately each algorithm has its own strengths and weaknesses. RHC and SA are good for fast computation problems with simple structure where the input space can be traversed. But GA and MIMIC really shine for problem with more complicated structures which need a good understanding of the underlying distribution.