INTRODUCTION

The reader can expect the following structure in this paper. First, I provide a brief description for each of the four algorithms (Random Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA) and MIMIC) used in this assignment. Here I also do some comparison and discuss operating considerations. This is helpful in laying the foundation for discussion in part 2. Next, I go over the process used to swap back propagation in the neural network constructed in assignment 1 for all these randomized optimizers (except MIMIC). Specifically, I use the Titanic data from assignment 1 for part 1. Finally, for part 2, I review the results of running and tweaking all these randomized optimizers on a number of classic data sets.

RANDOMIZED OPTIMIZATION

Randomized optimization refers to a class of algorithms which use randomization to maximize a fitness function. A fitness function, also known as an objective function, maps inputs in the input space to a score. The goal of the optimization is to find the input, which when passed to the fitness function, produces the maximum result.

Randomized optimization can be used to optimize the weights of a neural network which I do below for part 1. In this case the score being maximized is the negation of the error and the fitness function maps this score to the weights of a neural network.

Generally, optimizers can use different approaches to solving a problem depending on certain assumptions. For example, the Generate and Test approach iteratively passes values selected from the input space to the fitness function and evaluates which input produces a maximum value. This approach works best with a small input space and may useful when the fitness function is complex. If the fitness function has a derivative and can be solved to 0, then a calculus approach can be used. If the derivative of the fitness function is not solvable to 0, but has a single maximum, then Newton’s method can be used. Randomized optimization many be useful in the case where there is big input space or the fitness function is complex having no derivative and many of local maxima.

Hill climbing describes a class of optimization algorithms which-given some starting point-move in the direction of improvement as determined by a neighbor function until an optimum is discovered. Using random restarting-rerunning the hill climber some number of times from randomly selected starting point-reduces the risk of getting stuck at a local optimum. Random restart hill climbing may quickly identify a global maximum if its basin of attraction is large, meaning many points lead to the global optimum by adding stopping criteria to handle the case where many random starting points result in the same result. Another performance optimization is to keep track of previously visited points so that worst case, all points are visited once.

Randomized hill climbers assume moving towards improvement is best; they exploit this information. In contrast, simulated annealing-another randomized optimizer-favors exploring the space, or search. There is a tradeoff between exploring and exploiting which is reminiscent of the classic bias-variance tradeoff. Like a high bias, underfit model, an algorithm that only explores does not listen to the data, and could only happen upon an optimum randomly. Conversely, like a high variance, overfit model, an algorithm that only exploits may listen to the data too closely and get stuck at a local optimum.

Simulated annealing works by calculating a probability of whether or not it should jump to a new point in the broader neighborhood. The factors influencing this decision are how much improvement the new point offers and a temperature factor. The temperature factor-so named because annealing is a metallurgy term meaning to heat and then let cool slowly-determines the willingness to move. When the temperature is high, there is more willingness to explore and move to points with no improvement. When temperature is very low, there is less willingness to explore and the algorithm behaves like an ordinary hill climber. Just like with metallurgical annealing, the temperature factor typically decays slowly. This allows the algorithm to explore the input space early, ideally landing in the basin of attraction for the global optimum.

Genetic algorithms are another class of random optimizers. These algorithms have a random component that explores and a mutation and crossover mechanism that exploits. The term genetic is meant to draw comparison to the theory of evolution which says that random selection and mutation can explain populations of similar things. Following this analogy, points are called individuals and each subsequent iteration is referred to as a new generation.

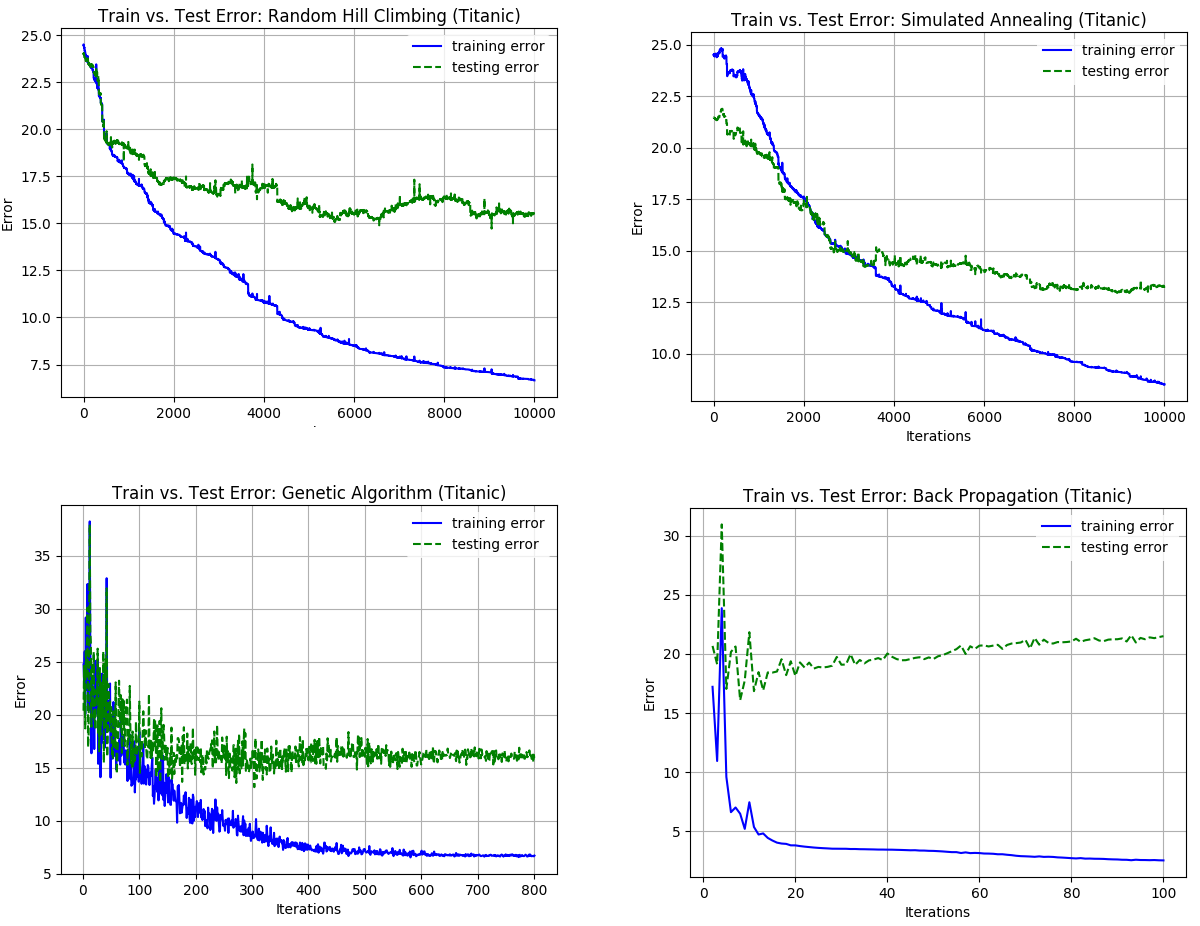
The genetic algorithm scores all individuals based on the fitness function and selects a subset using some selection criterion. In the case of truncate selection, it is completely based on exploiting the fitness score. In the case of roulette wheel selection, randomness is injected to explore the data. The individuals not selected are removed and replaced with new offspring created by crossover, which is a generic term describing the process of combining attributes of individuals to form a new individual. Mutation may be used during crossover to randomize the process. There are a number of crossover techniques. Uniform crossover is most analogous to genetics in that each gene comes from either one parent or the other. This differs from one-point, two-point and cut and slice techniques which swap segments of genes.

The MIMIC algorithm operates differently than the other randomized optimizers discussed so far in that it involves a building a probability distribution of the points. This distribution is a way to retain information in the data, called structure. This differs from other randomizers which may restart the next iteration with a blank slate. These algorithms operate similar to genetic algorithms in that the best fitting points are used to estimate the probability distribution for the next iteration. Best fit in this case means points have a fitness score greater than a theta value. Although there are different ways to estimate a probability distribution, the most basic-while also capturing relationship information-is a dependency tree. This can be done by calculating the maximum spanning tree. One additional benefit of the dependency, besides being simple, is it is easy to sample which is done a lot at each iteration.

Compared to other randomized optimizers, MIMIC tends to outperform when structure is present in the data. One can think of structure like a pattern, a sign wave for example. Although it is better than some other randomized optimizers when it comes to avoiding getting trapped in a local optimum, it is not immune to this problem. Empirically, MIMIC converges in far fewer iterations than other randomized optimizers, which makes it attractive for time consuming fitness function, like complex simulations or human evaluation. But one drawback of MIMIC is that, unlike simulated annealing and randomized hill climbing which only look a few samples per iteration, it pulls many more samples per iteration; this requires much more compute time per iteration.

COMPARISON TO BACK PROPAGATION

For part 1, I exported already scaled and split Titanic data from the first project for use in ABAGAIL. All categorical values were converted to binary using the same one hot encoding performed for assignment 1. I created a new class modelled after opt.test.AbaloneTest to import Titanic data, and calculate and save training and testing error at each iteration. I made sure the number of input and hidden layers matched the optimal configuration found during project 1. This worked for randomized hill climbing, simulated annealing and the genetic algorithm. I also used ABAGAIL to run back propagation by creating a new class modelled after func.test.NNClassificationTest and modifying the shared.Convergence class to calculate and save training and test error at each iteration. Finally, I imported the errors into python to generate the plots seem below.

 I ran each model up to 10,000 iterations, but in the case of the genetic algorithm, only plotted up to 800. Similarly, back propagation required far fewer iterations to identify the pattern.

INTERESTING DATASETS

As the name suggest, the Four Peaks (FP) dataset has a total of 4 maxima, 2 of which are global maxima and 2 of which are suboptimal. There are two configurable parameters for this dataset. The total number of inputs is the N value and a T value which Isbell sets to a constant of 10% of N in his paper.

The Travelling Salesman Problem (TSP) tries to find the shortest path between cities subject to two constraints: 1) every city is visited exactly once and 2) the route must conclude at the origin city. This is a classic problem, frequently studied in theoretical computer science. The number of cities is configurable as the N parameter.

The Max K-Coloring (MKC) problem is derived from notion of K-Colorable which describes a graph having no adjacent nodes with the same color. Determining if a graph is K-Colorable is NP-Complete meaning no known fast solution exists and the time required to solve the problem increases exponentially as data increases. MKC attempts to minimize the number of adjacent nodes having the same color. There are 3 configurable parameters for this dataset: 1) number of vertices in the graph 2) the number of adjacent nodes per vertex 3) number of available colors.

The Knapsack dataset models the problem of placing object in a knapsack. Each object has a weight and a profit associated with it and the goal is to maximize total profit subject to a constraint on total weight and of course, total volume of the knapsack. There are 5 configurable parameters for this dataset: 1) number of objects available, 2) maximum number of copies of each object, 3) maximum weight of any object 4) maximum volume of any object and 5) a size of the knapsack relative to the max volume available in the input space.

