Neural Network training through Randomized Optimization

The Skin dataset is a set of RGB points taken from portraits of human faces. Each point is labeled skin or non skin. In assignment one, I showed that neural network trained using backpropagation of error can learn a function that labels points very accurately. In part one of this assignment, I train the same neural network using a variety of randomized optimization functions, including randomized hill climbing, simulated annealing, and a genetic algorithm.

Randomized hill climbing begins training by selecting a starting set of weights as a random point near the origin in the space of the neural network weights. It then samples weights in the neighborhood of the current instance up to a limit that I call the neighborhood size. It moves to the first point that improves upon the fitness of the current instance and begins sampling again, or terminates if no point in the neighborhood can improve on the current point as it has reached a peak. Three functions, `random`, `fitness` and `neighbor` must now be defined.

The `random` function returns a set of random weights between -0.1 and 0.1. Small weights are preferred when training neural networks because large weights can overcome features of the data and fail to learn the target function. Tests with random weights scaled by factors of 1 and 10 were found to be less suitable than the 0.1 scale.

The `fitness` function takes as input an instance of network weights. Its output must be real valued and increasing as the suitability of the network weights to the classification task at hand increases. To accomplish this, I initially select a small training set at random from the full skin dataset. Each time `fitness` is called it finds the output of the network given the input weights over the training set and outputs the count of the correctly classified instances, less the margin in the wrong direction of misclassified instances. This definition allows fitness to increase by getting more classifications right, or by getting more classifications less wrong.

The `neighbor` function takes an instance of network weights and returns an instance of network weights that are nearby on some distance measure. Exhaustive search in the neighborhood of all orthogonal unit vector offsets from the current point finds lower peaks than sampling in random directions because peaks between steps are never reachable with fixed units and orthogonal directions. Sampling random neighbors require a limit on the sample size, as there are an infinite number of neighbors that are one random vector away from the current point. This `neighbor` function returns a new set of network weights that is the sum of the given weights and a new set of weights returned by `random`.

The rest of the implementation of randomized hill climbing was a control loop to train the neural network until stopping criteria are met. Here, training stops when no point in the neighborhood can improve on the fitness of the current point more than a minimum improvement parameter set to 0.01.

Figure 1. Fitness of Randomized Hill Climbing on skin classification problem

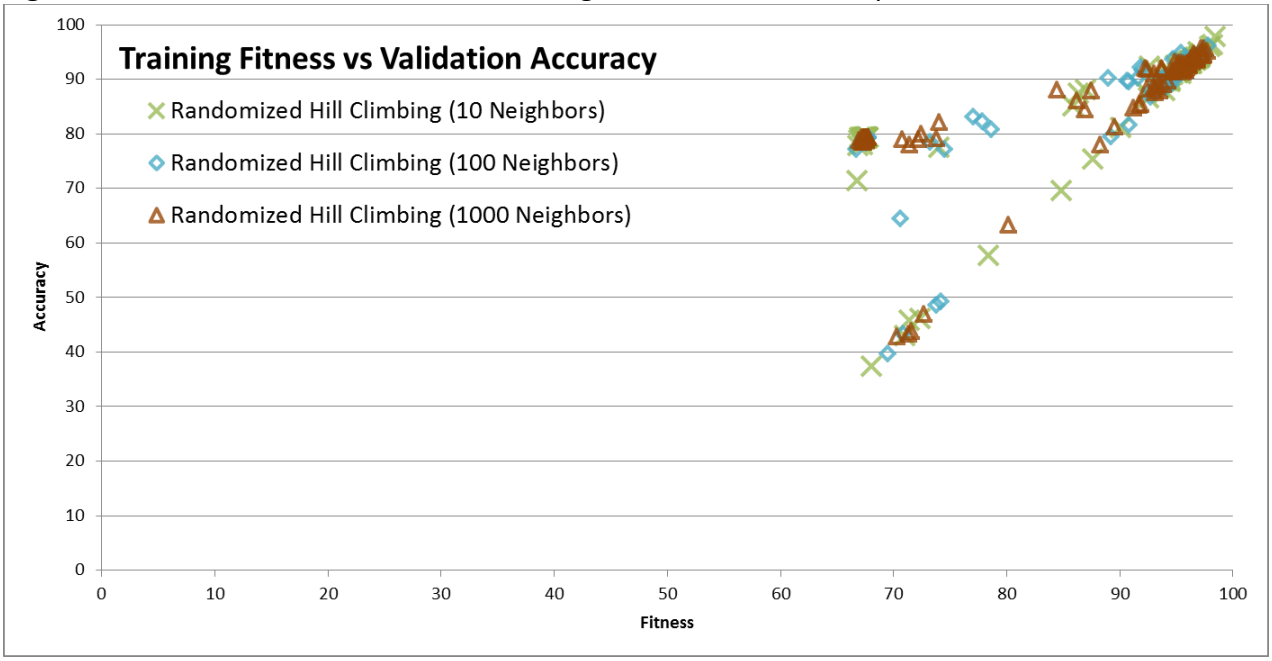

Table 1. Results of 100 Randomized Hill Climbing starts on skin classification problem

Neighborhood Size	Training time for 100 starts (sec)	Optimum fitness	Validation accuracy of optimum (%)	Standard deviation of fitness	Minimum fitness
10	28.518	98.46	97.75	13.31	66.82
100	28.197	97.85	96.07	13.01	66.70
1000	28.065	97.74	95.17	12.74	66.96

The training time did not increase while increasing the number of samples tried before stopping. Increasing the neighborhood size parameter decreases standard deviation of fitness slightly, which decreases slightly the expected number of random starts before a suitably fit optimum is found with a certain probability. The validation accuracy of the most fit result decreased as neighborhood size increased, which implies that larger neighbor sample sizes lead to overfitting the training data.

Other variables that were not varied include the size of the training set, the minimum fitness improvement criteria, and the scale parameter used for the random step. Randomized hill climbing is known to be prone to stopping in local optima that are less fit than the global optima. Simulated annealing overcomes this limitation with a temperature parameter that allows it to randomly step out of local optima while hot, and climb to the nearest peak when as it cools.

Simulated annealing can be built on randomized hill climbing with a few modifications. Instead of stopping criteria based solely on exhaustive neighborhood search, there is a random component that will accept a less fit instance from time to time. The temperature parameter is

used to control the selection of the next instance, such that the next step is nearly random when temperature is high, but seeks fitness improvement when the temperature falls. A cooling parameter controls the rate of temperature decay with each step.

The formula used for the probability of accepting a less fit instance is taken from the ABAGAIL implementation `SimulatedAnnealing`. The class itself is not used here; the encapsulation of temperature and exposure of a single train() method made it difficult to take the current temperature into account in the stopping criteria. Instead, the algorithm is implemented with the ABAGAIL `RandomizedHillClimbing`, and the outer control loop takes into account temperature and adds the random step component.

For the starting temperature, I arbitrarily chose to use one half of the maximum double precision floating point value in Java. A cooling factor of 0.1 was chosen after some experimentation. The neighborhood size of 10 was retained from the previous experiment.

The fitness of weights found by simulated annealing improves on optima found by randomized hill climbing, and the validation accuracy did not suffer. The standard deviation of fitness was cut in half, but training time doubled. Most notably, there are fewer misfit optima with high fitness but low validation accuracy found using simulated annealing. Whether this is a feature of the problem space, or a feature of the algorithm remains to be seen. If the increase in time complexity is affordable, or the cost of random starts is sufficiently high, then simulated annealing is preferable to randomized hill climbing.

The third optimization algorithm tested was the ABAGAIL implementation `StandardGeneticAlgorithm`. The algorithm maintains a population of instances, and randomly mutates and mates them each generation, replacing the least fit with offspring of the fittest. The parameters are population size, offspring per generation, and mutations per generation. Two new functions have to be defined for neural network weights, a mating function and a mutation function. The `mate` function takes two instances of weights and returns a new instance that is some combination of the two. The `mutate` function modifies the weights of the given instance. Here the offspring of a pair is created from the average of their weights, and the mutation function adds weights from a random instance to the given instance.

The parameters chosen impact how the population develops. Each mating will result in a new instance between two other successful instances, and each mutation will result in a random step. After some experimentation, I elected to mutate half of the population each generation, and mate five times per generation. More mutation might lead to faster convergence, or possibly even divergence; more mating might lead to slower search and a more homogenous population.

The runtime for the standard genetic algorithm is significantly slower than hill climbing or simulated annealing. This makes sense given that each generation must mutate and mate numerous instances, where hill climbing only had to mutate once per iteration. The advantage to this is the population explores many paths at once, moving by random steps and jumping from less profitable paths to explore the midpoint of two profitable ones. The downside is that

there is no pressure for individual members to improve through search, and only the periodic replacement through mating will prune the poor performers that have wandered into low fitness space.

The stopping criterion for the genetic algorithm's control loop is a fixed number of generations. I varied the number of generations from 100 to 300 in increments of 100. Training time scaled linearly with the number of generations trained. The fittest member of each population after stopping was measured on the training and validation sets.

As the generations increase from 100 to 300, the fitness and accuracy approach the peak found by simulated annealing, but does not quite reach the same optimum performance. Additional iterations would likely allow the algorithm to improve further, but the time cost discourages the selection of this genetic algorithm over simulated annealing to find absolute optima. Another possible solution is to initialize a population of instances with this genetic algorithm as exploration, then train each member with a hill climbing algorithm to find the nearest peak

Figure 3. Performance of optimization algorithms on skin classification

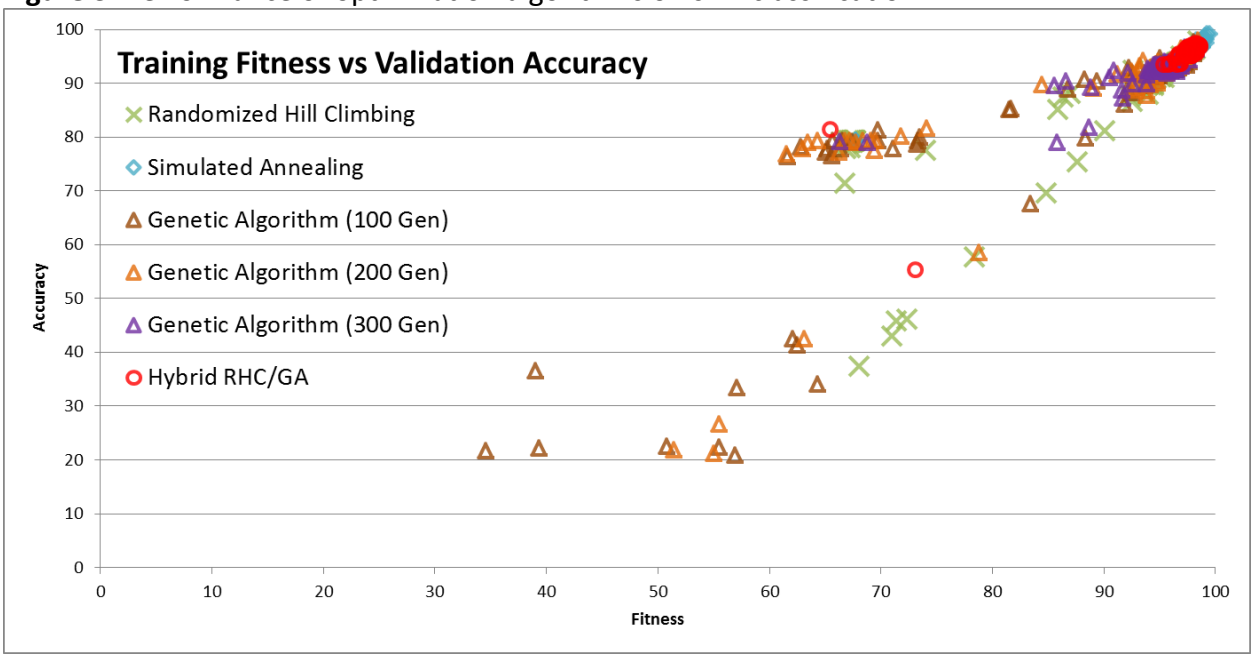

Table 3. Results of optimization algorithms on skin classification

Optimization algorithm	Training time (sec)	Optimum fitness	Accuracy of optimum	StDev fitness	Minimum fitness
Randomized hill climbing	28.518	98.46	97.75	13.31	66.82
Simulated Annealing	57.432	99.53	99.04	6.61	67.75
Genetic Algorithm	782.300	98.27	96.30	4.52	66.32
Hybrid RHC/GA	54.959	98.63	96.97	4.09	65.44

The simulated annealing algorithm found the best weights for our skin classifier in terms of fitness and validation accuracy. The standard genetic algorithm found optima with least variance across starts, although it did not hit the peak of simulated annealing. A hybrid genetic algorithm followed by hill climbing performed in similar time to simulated annealing and similar standard deviation to genetic algorithms, but failed to find a global optima comparable to the best found by simulated annealing. This could be chance, or it could be that hill climbing and genetic algorithms were both attracted to a particular region of optima that simulated annealing was able to escape to find a better solution.

Contrast of optimization algorithms over selected problems

The flip flop evaluation function counts the number of alternating bits pairs from right to left until the first non-alternating consecutive bit pair is encountered, then returns the count. Hill climbing and simulated annealing should have no trouble optimizing this problem, given enough iterations, since for any given instance, there is exactly one neighbor within a single bit flip that improves the function. A genetic algorithm with a single bit mutation function and a single point crossover function should also be able to find the absolute optimum eventually, since each mating has a good chance of preserving an alternating string of bits to the right and potentially extending that string to the left from an otherwise unfit instance. The MIMIC algorithm should be able to learn a dependency tree distribution that captures the alternating structure of bits from right to left. Any flat uniform distribution with no conditional dependence could not learn to optimize the flip flop function, because the alternating bit pattern of the optima averages to the uniform distribution.

The four peaks evaluation function counts the consecutive leading ones in the head, the consecutive trailing zeros in the tail, and then awards a bonus equal to the size of the bit string if both the head and tail are longer than the parameter t. This is tricky for hill climbing algorithms, because for any given instance there are two neighbors that can increase fitness, but if the head and tail do not both grow to t before the algorithm stops, or if one of them grows beyond length minus t, then the largest fitness boost will never be found. Simulated annealing might allow more time to find the balanced head and tail in some iterations, but it will still tend to find lesser optima. Genetic algorithms have a better chance of finding the bonus through a single point crossover. MIMIC should excel at this, once one sample finds the bonus, it will be preserved in the kept pool to train the optimal distribution for all remaining iterations.

The continuous peaks evaluation function is like the four peaks function in that it has a parameter and it counts runs of ones and zeros, but unlike the four peaks function, it counts the longest run of ones and longest run of zeros anywhere in the bit string. This provides more ways to improve fitness in any given iteration by flipping a bit on either side of the longest run of zeros and the longest run of ones, rather than on the inside of either the head or the tail as in four peaks. That difference makes it easier for hill climbing algorithms. It should not change the fitness of genetic algorithms or MIMIC to optimize this problem.

The count ones algorithm does what it says. It should be trivial for the hill climbing based algorithms to find a neighbor that improves any given instance in the early iterations, but once the zero positions become sparse, further improvements become harder to find. Genetic algorithms should perform well, because as they replace the poor performers with crossovers of good performers, zeros should be selected out. The MIMIC algorithm should also quickly approach an optimum distribution of all ones, but the probabilistic nature might still return a suboptimal result from the optimum distribution.

Each test used the same criteria used by the ABAGAIL test classes, and was repeated 10 times to get a distribution of optima found by the implementation. Randomized hill climbing and simulated annealing were each allowed 200,000 iterations to find a peak. The standard genetic algorithm was allowed 1,000 iterations with a population size of 200, a mating rate of 100 per generation and a mutation rate of 10 per generation. MIMIC was allowed 1000 iterations of 200 samples, keeping 20 per iteration, and configured to use a discrete dependency tree for the learned distribution. In each case the experiment is designed to be roughly equivalent to 200,000 iterations of randomized hill climbing.

Table 4 Optimization Problem Results

Problem	Algorithm	Total Time	Min	Max	Median
Flip Flop	RHC	00:01.4	61	69	65
Flip Flop	SA	00:01.9	78	79	79
Flip Flop	GA	00:01.5	65	74	71.5
Flip Flop	MIMIC	01:04.0	69	76	73.5
Four Peaks	RHC	00:01.2	80	80	80
Four Peaks	SA	00:01.5	80	151	115.5
Four Peaks	GA	00:01.4	91	109	97
Four Peaks	MIMIC	01:04.4	51	151	136.5
Continuous Peaks	RHC	00:01.2	72	106	83.5
Continuous Peaks	SA	00:01.6	85	112	105
Continuous Peaks	GA	00:01.6	79	93	86
Continuous Peaks	MIMIC	00:40.3	93	111	100.5
Count Ones	RHC	00:01.0	60	60	60
Count Ones	SA	00:01.4	60	60	60
Count Ones	GA	00:01.4	59	60	60
Count Ones	MIMIC	00:40.2	60	60	60

The maximum possible fitness for the flip flop function given a bit string length of 80 is 79. Simulated annealing consistently found the global optima. MIMIC never returned the global optima, and took 33 times as long to arrive at a result. The genetic algorithm performed almost as well as MIMIC, and was faster than simulated annealing. Randomized hill climbing is dominated by all others in this application, failing to find any optima more fit than the least optimum found with other methods.

The maximum possible fitness for the four peaks given a bit string length of 80 and a peak size of 8 is 160. Randomized hill climbing consistently found the local optima that just maximized the head or tail, never meeting the peak criteria. Simulated annealing, MIMIC and the standard genetic algorithm met the peak criteria more often than not, but none of them found the global optimum in the iterations allotted. MIMIC and simulated annealing found equivalent maximum optima but MIMIC's median optima were better than those found by simulated annealing. MIMIC has a better chance of keeping instances that meet the peak criteria to train its distribution once they are discovered, and then reproducing something resembling them in sampling. Simulated annealing merely has a means of escaping the basin of attraction of the local optima that always trapped randomized hill climbing, but chance determines whether the hot random walk will meet the peak criteria and become attracted to the global optimum before the cooler fitness improvement phase takes over.

The maximum possible fitness for the continuous peaks problem given a bit string length of 60 and a peak size of 6 is 120. It is very similar to the four peaks problem but the criteria for fitness are relaxed as the calculations from the lengths of leading 1s and trailing 0s are replaced by the lengths of the longest consecutive run of 1s and longest consecutive runs of 0s. All algorithms were able to find optima that met the peak criteria. Simulated annealing and MIMIC both performed very well, but failed to find the global optima. Simulated annealing found better maximum and median optima than MIMIC. Randomized hill climbing found a better maximum optima than the standard genetic algorithm, but the latter had less variance and a higher central tendency.

If `ProbabilisticOptimizationProblem.getOptimal()` returned the maximum likelihood instance from the learned distribution rather than a sample from the learned distribution, that it would better represent how close MIMIC came to the global maximum.