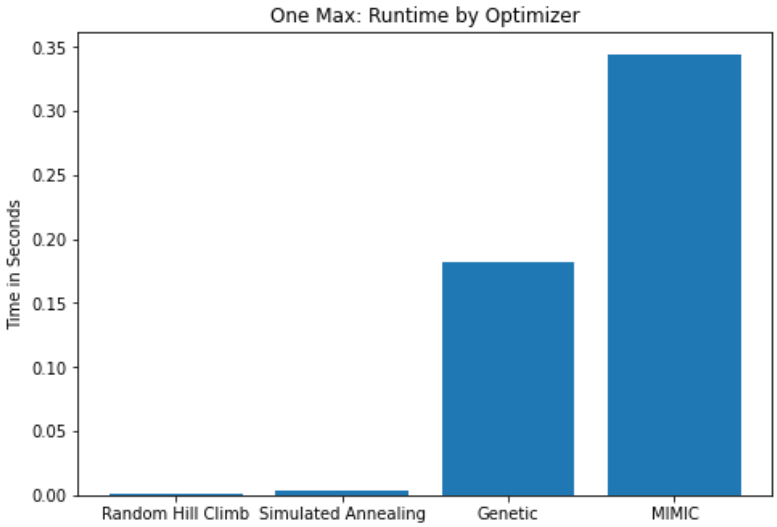
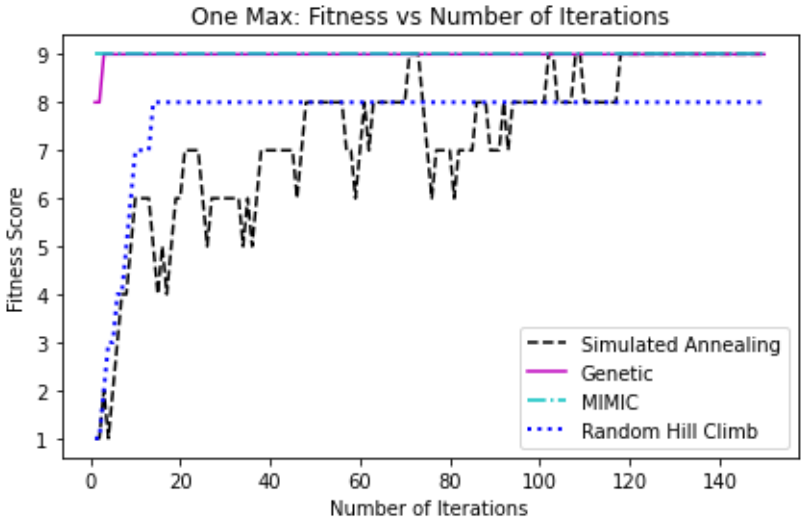
Introduction

I implemented the four optimization algorithms using the python library mlrose. I also used the same library to analyze using random hill climbing, genetic, and simulated annealing to calculate the weights in my neural network.

One Max: Strengths of Genetic Algorithm





The goal of the one max problem is to find a bit string that has the maximum sum of all the bits in the string. This happens when all the bits in the string equal one. This problem was chosen to show the strengths of genetic algorithms. The one max problem can be solved with successive bitwise combinations, something that was a feature of the motivating example for genetic algorithms that was given in the lecture. In this implementation, a random population of bit strings is generated. Two parents are selected from the population. A random position n is chosen in the string so that the child has all elements before position n of parent 1 and all elements of parent 2 after n. There is also a mutation parameter where the bit string elements each have a 10% chance of being mutated, which here means that 1 would be 0 and 0 would turn to 1. It is only after 1 time of going through this process that the entire random population's overall fitness was 8. Only 3 iterations were needed to reach the max fitness of 9. This shows how much info can be transferred from the parent generation to the next generation within one iteration. This also makes it so that finding the maximum fitness is independent from the initial state that is passed to the optimization algorithm, which is a weakness of simulated annealing and random hill climbing.

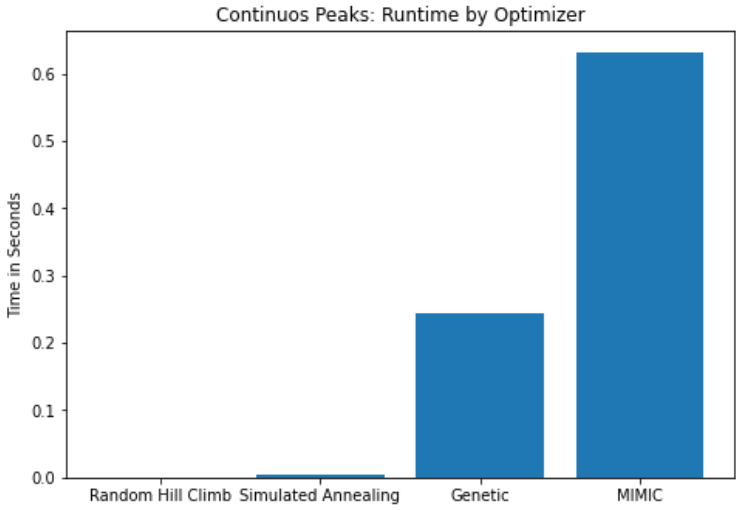
MIMIC, like genetic algorithm, also performs well. Only 1 iteration is necessary for finding the optimum bit string. Out of the 4 algorithms, it takes the longest to run.

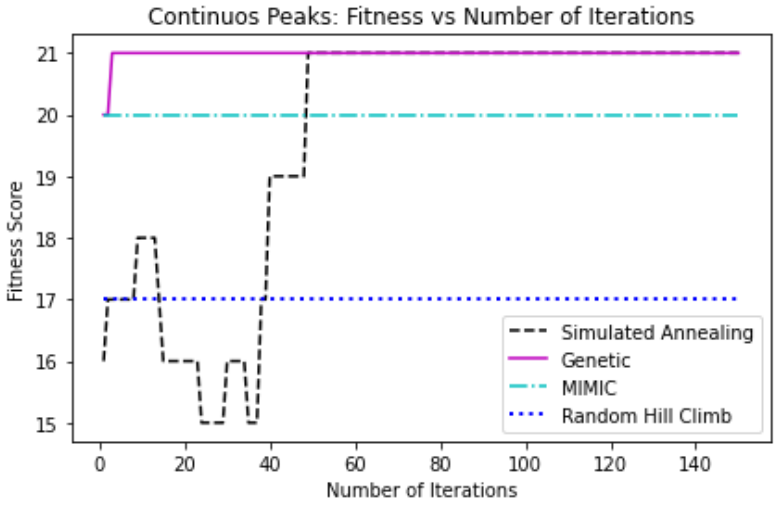
A weakness of simulated annealing is clearly shown here in the fitness versus iteration graph. At several points, simulated annealing finds the true optimum but instead of stopping it continues to search for a solution with even higher fitness. The fitness goes down with more iterations before eventually levelling off at around 120 iterations. The cooling is advantageous

when working with a function like continuous peaks, but it can lead to unnecessary searching through the fitness space like it does here in the one max problem.

Random hill climb does not even reach the optimal bit string with the highest fitness. This shows random hill climbing's propensity to get stuck at a local optimum without finding the true optimum.

Continuous Peaks: Strengths of Simulated Annealing



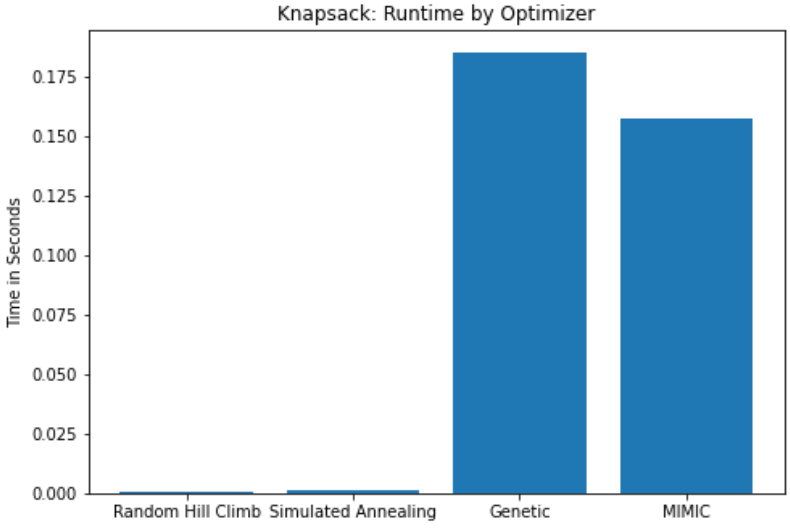


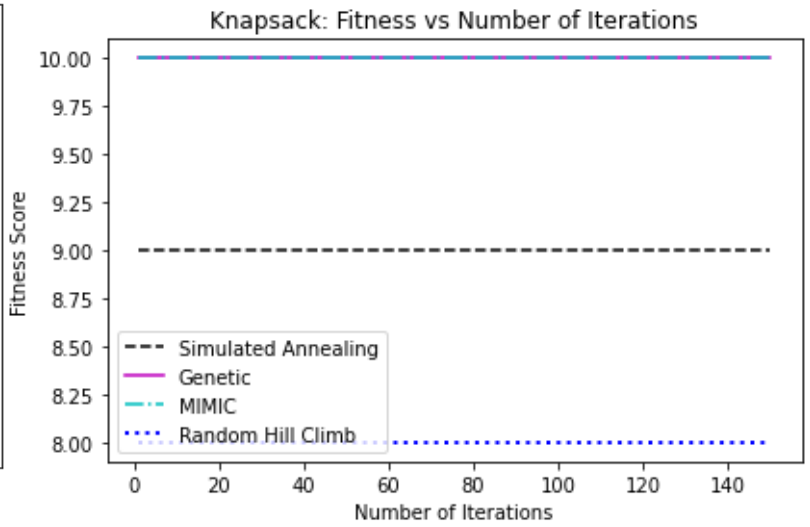
The goal of the continuous peaks problem is to find the max value of a function with many local optima. This problem was chosen to show the strengths of simulated annealing. This function has the kind of pitfalls that are ill addressed by random hill climbing. There are many local optima which makes it highly likely that random hill climbing will get stuck at one of the local optima and not find the global optimum of the curve. Random hill climbing here also depends highly on the initial state given to the algorithm. If you are lucky, the initial state is close to the optima and will climb in that direction but that is an uncertainty. In the graph you see that the random hill climbing curve is flat. The initial state I happened to give the algorithm is a local optimum so in either direction, the value of the fitness goes down so random hill climbing will not allow a search past the optima.

Simulated annealing on the other hand has a built-in mechanism to better explore the curve and climb past the smaller hills and get to the true optimum of the curve. Although the initial state is important for later performance, the simulated annealing ability to go in a negative direction while searching for another peak means it is less likely to get stuck. This is clearly shown in the graph where when given more iterations, the fitness goes up and then actually goes down but after around 50 iterations levels off to the true optimum.

MIMIC and genetic algorithms also perform well on the continuous peak problem. They both with very few iterations, MIMIC 1 and genetic 2, reach the global maximum. Interestingly, MIMIC does not reach the true optimum here. Genetic does but it takes much longer than random hill climbing or simulated annealing to do it.

Knapsack: Strengths of MIMIC





For the knapsack problem, you have a set of objects that each have a weight and a knapsack that can only hold a fraction of the total weight of all the objects. The goal is to take as many objects as possible without going over the weight threshold of the knapsack. This problem was chosen to show the strength of MIMIC. MIMIC and genetic algorithm are the only two of the four algorithms that reach max fitness.

MIMIC, like genetic algorithms, starts off with a random sample population of bitstrings. It then selects the strings with the top probability density above a threshold n. This becomes the new population until the best one is found. MIMIC solves this problem well because the underlying structure of the knapsack function is unimportant to find the solution. The probability density of each bit string can be calculated. And you only need one iteration to get to the best fitness. In this case, MIMIC took less time to reach the string with the highest fitness than genetic did.

Genetic algorithm does well here, again because it is a combinatorial problem and with a short string length, the initial random population already has the optimal bit string contained in it. After the first iteration, that string would be selected.

Both simulated annealing and random hill climbing are stuck. A single change in the bit string leads to large drop in the overall fitness that is too negative for random hill climbing to go in that direction and too negatively large for simulated annealing that it would lead to a large temperature drop and not the smaller one that simulated annealing is more likely to accept and move in the direction of.

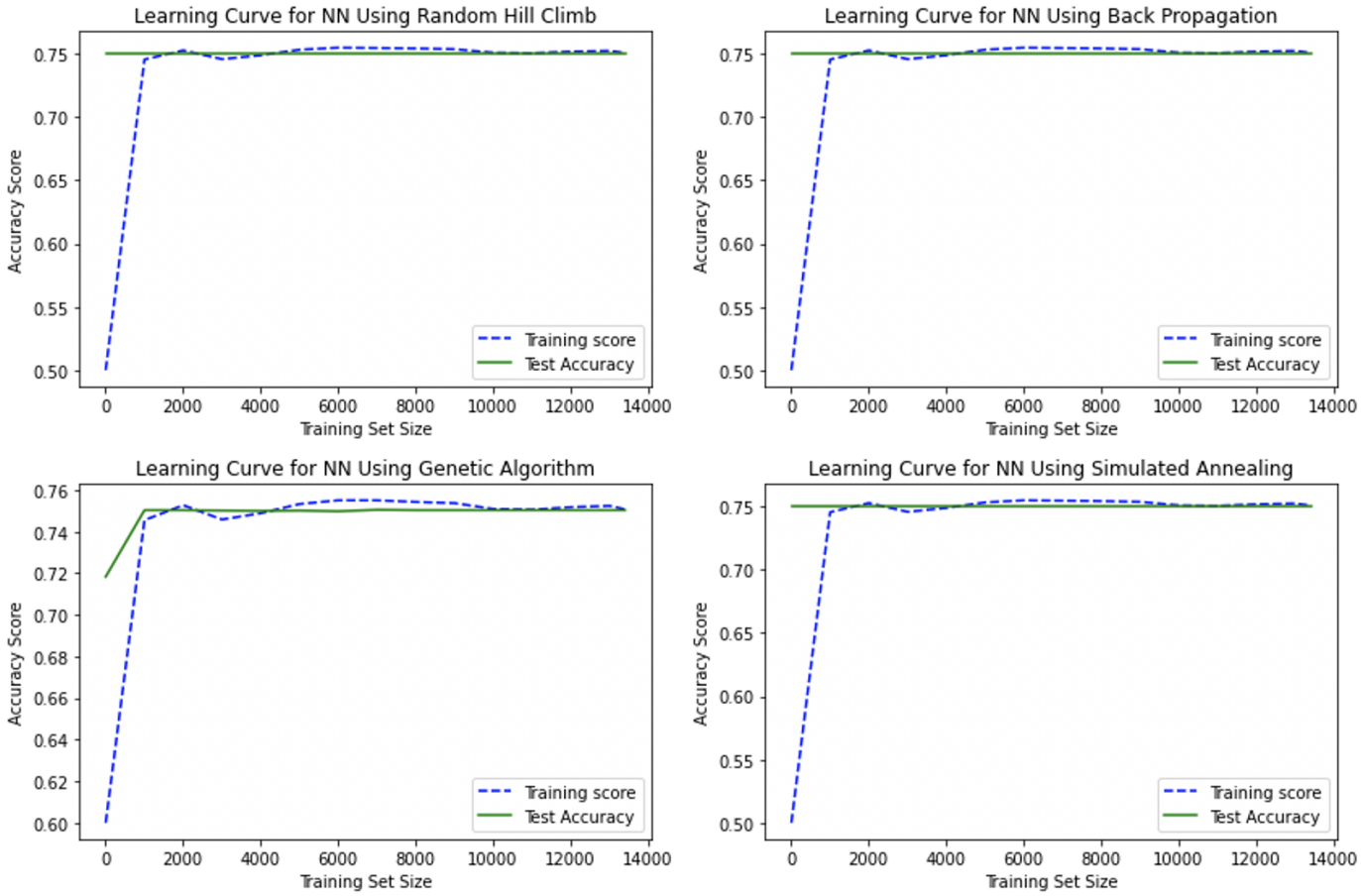
Neural Network Weight Optimization

I chose to use the data set that represented the job churn of data scientist for analyzing weight optimization for neural networks. The target concept is whether an employee who took a

course offered by a company wants to continue to work for the company after course completion. The predictors include demographic, education, and job experience.

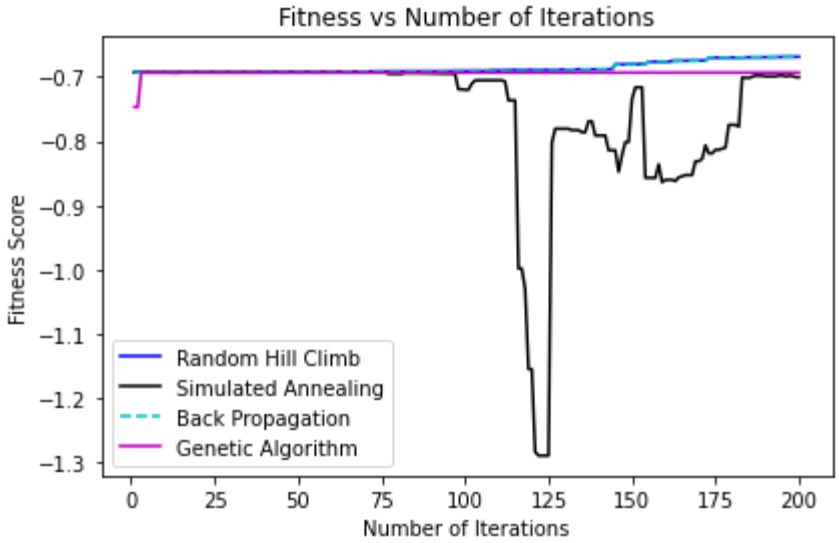
In the first assignment I used the MLP classifier in the scikit learn library. Since the loss and activation functions were different between the scikit learn library and the mlrose library, instead of using scikit learn like I did in the first assignment, I used. The default backpropagation neural networking the mlrose library so that I was comparing comparable things. I did create a more accurate classifier in the first assignment using the same data set but it was hard to tell why that was. Was it because of how many hidden layers I allowed? Was it because of the activation and loss functions? I just avoided that confusion by not using it at all.

In the other optimization exercises, we analyzed problems with discrete outputs with limited dimensionality and limited random neighbors in that there are only a finite number of states that could be considered neighboring the current state. In the case of neural network weights, the dimensions are much higher, and the outputs are continuous, and a random neighbor is potentially infinite. Simulated annealing and random hill climbing select their next move by choosing a random neighbor and only moving if the cost function at the neighbor meets the respective requirements for moving. This is the same method that the discrete output is using but now it's just randomly searching a larger space. Genetic algorithm creates a finite subset of the weights, in this case 200, and searches for the best out of this subset. In the simple discrete problem with small bit strings, it was almost guaranteed that the data set of 200 would contain the maximum fitness but in a continuous space that is not guaranteed.



The learning curves for the training set size vs accuracy score was the same for simulated annealing, random hill climb, and back propagation weight optimization. These 3 algorithms can act very similarly or equivalently under certain conditions, so this we know as discussed in the lecture, the lower the temperature is, the more simulated annealing acts like random hill climb. Random hill climbing works by calculating the cost function in every direction; gradient descent works by calculating the slope of a neighbor. If the slope and the cost function are both steepest at each neighboring point, then gradient descent and random hill climb will have similarly. With all this in mind, that makes as to why these learning curves look the same for these 3 algorithms.

	Optimizer	Runtime	Test Accuracy	Train Accuracy	Weight Fitness
0	Simulated Annealing	1.245349	0.750348	0.750783	-0.692924
0	Random Hill Climbing	1.040527	0.750348	0.750783	-0.66848
0	Back Propagation	1.376907	0.750348	0.750783	-0.66848
0	Genetic Algorith	140.711315	0.750348	0.750783	-0.693147



Genetic took the longest by far but it took the shortest number of iterations to reach the best weights. This is expected behavior given what was seen in the optimization problems I explored earlier. It still had the same accuracy on the test set as the other three algorithms

Even though the best weights found by each optimizer were different and the overall fitness of the weights was different, the accuracy of the resulting neural network classifier that they built were the same. Since this is the case accuracy of the resulting neural network was the same for all four optimization methods, I would choose the best method in terms of runtime and fitness of the optimized weights. The default backpropagation method or random hill climb, which in the mlrose library are functionally equivalent although the underlying code for them is different. The fitness of their weights is the highest and they performed just as accurately on the test set as the other weight optimization methods.