CS 7641 Machine Learning
Assignment 2

Philip Bale
pbale3

Due Sunday March 11th, 2018 11:59pm

Part 1: Neural Network Optimization

Introduction

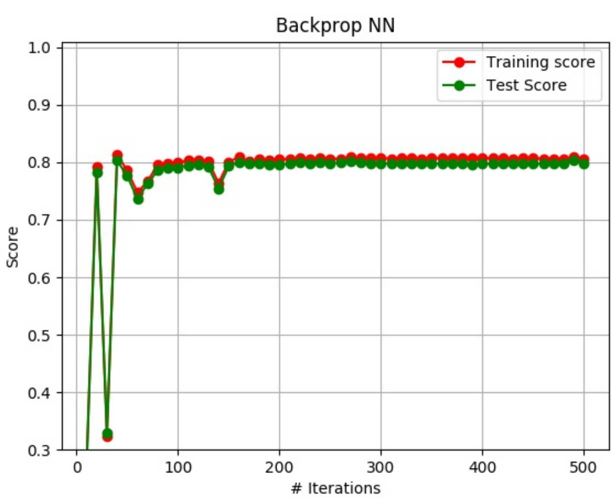
Part 1 of the assignment surrounds using randomized optimization to find the best possible weights for a specific neural network. In assignment 1, backpropogation was used to find optimal parameters for a neural network. This neural network took in various input features for US permanent visa applicants and then attempted to predict the outcome of an application. After various tests, I found the optimal parameters of: 6-node input layer, one hidden layer with 100 nodes, one output node, and about 500 iterations.

I chose this problem because, as someone who has worked with a large number of first-generation visa holders and immigrants, I am extremely interested in building tools to help others to achieve the same. At the end of the day, the goal is it to try to determine the application result before time, money, and other resources are spent.

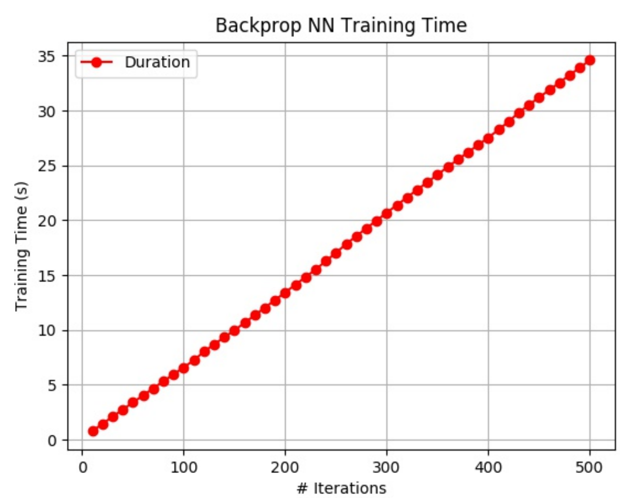
1) Backpropogation (Assignment 1)

Overview

The first weight-finding algorithm used was backpropogation. Backpropogation works by essentially calculating the error at the end of a network, and then working backwards to minimize that error over various iterations. An error (or loss) function is effectively minimized over time using this backpropogation technique. As discussed in assignment 1, the permanent visa is rather large and robust. It is quickly learnable by various different learners and in such backpropogation found significant success.



Permanent Visa Applicant NN Learning Curve



Permanent Visa Applicant NN Learning Curve

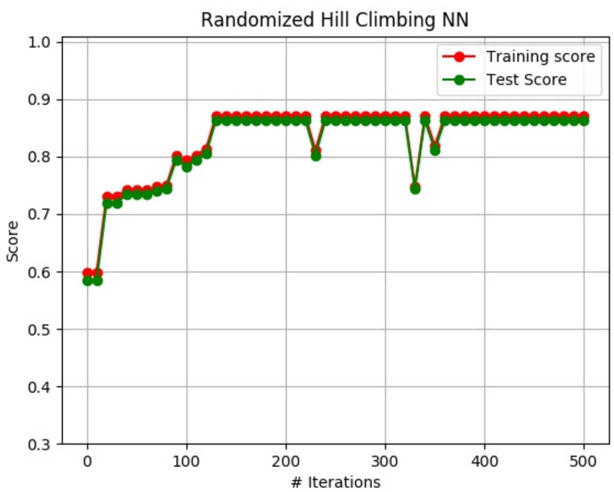
Right around 50 iterations, the network begins to converge at around an 80% successs rate. Seeing as the training and test score track either rather closely, it is apparent that the dataset is rather robust and consistent. One thing to note is that the training time scales linearly with the number of iterations–which makes sense since the same amount of calculations with similar complexity are performed on each iteration of backpropogation.

2) Randomized Hill Climbiing

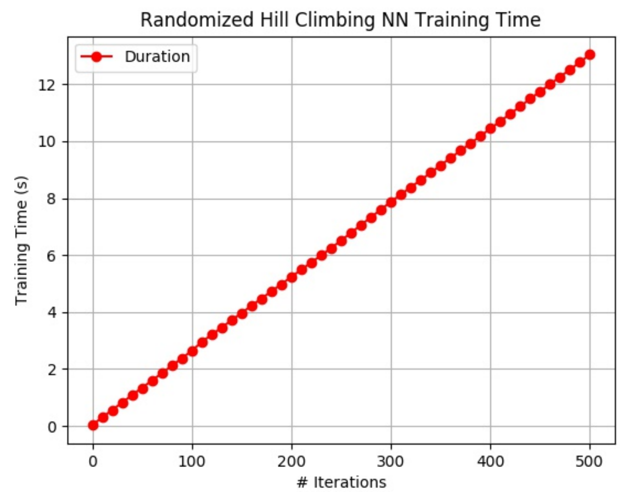
Overview

The second weight-finding algorithm used was randomized hill climbing. Randomized hill climbing works by taking a random starting point and then incrementally attempting to improve on that point. In the context of a neural network trying to find weights, randomized hill climbing selects random weights and then moves in a direction so as to try to find a better result for that weight–akin to trying to move up an optimization 'success hill'.

One thing to note is that we are using randomized hill climbing, not random restart hill climbing. In such, the algorithm is prone to getting caught in local optimizations, or local maximums.



Permanent Visa Applicant NN Learning Curve



Permanent Visa Applicant NN Learning Curve

High accuracy results are achieved right around 125 iterations+. While this is subject to randomness, by looking at the results it is shown to become rather consistant. By looking at the score results, one can see various instances of the randomized hill climbing getting caught in local optimas and being unable to escape. Such is the case around 220, 320, and 350 iterations.

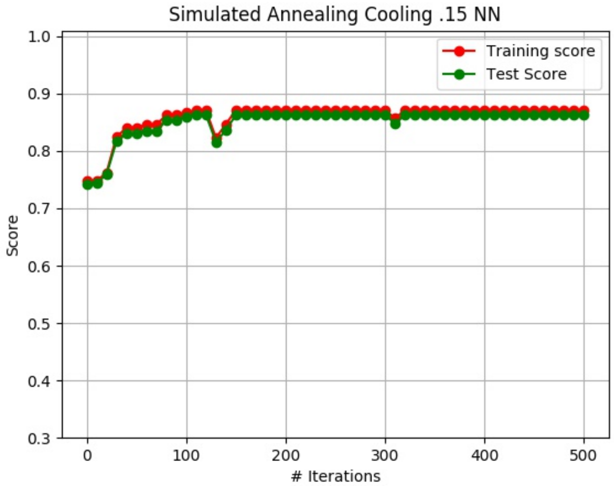
Similar to backpropogation, training time scales linearly with the number of iterations run. Training time tends to be a bit faster using randomized hill climbing because of a reducation in calculations necessary. Whereas backpropogation needed to do calcluations to minimize error moving backwards through the network, randomized hill climbing simply needs to move in one direction and determine if the new weights are better.

3) Simulated Annealing

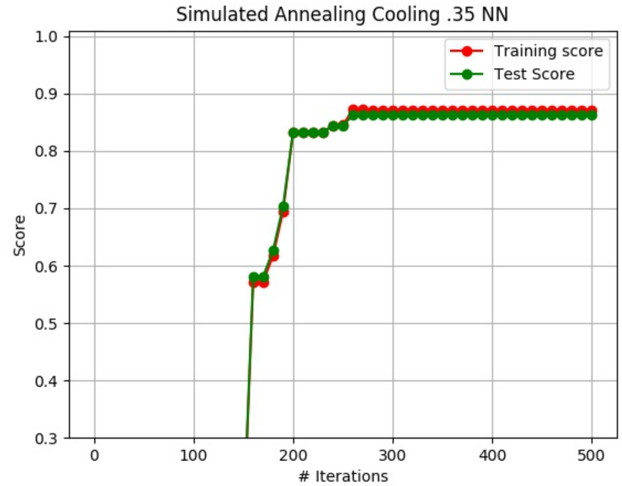
Overview

The third weight-finding algorithm used was simulated annealing. Simulated annealing works by taking a random solution and then samples nearby alternatives. By comparing the alternatives to the original solution, the optimizer decides to either stick with the original solution or move to the new one. Using a temperature and cooling parameter, the algorithm is more open to worse solutions at first but gradually moves towards only accepting better solutions.

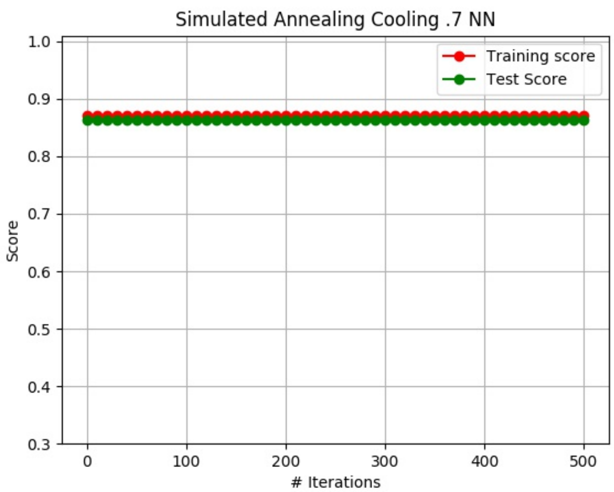
Various coolng parameters were used to try to determine the best simulate annealing approach. Below, success rates by number of iterations are showed for simulated annealing approaches with cooling parameters of .15, .35, and .7. The parameter of .15 and .35 take significantly longer to converge to the optimal solution than did a higher parameter. While part of this can be attribute to luck (the algorithm could have randomly found itself in an optimal solution earlier on)–part of this too is the way the algorithm operates by its parameters. The .7 cooling temperature here proved most effective.



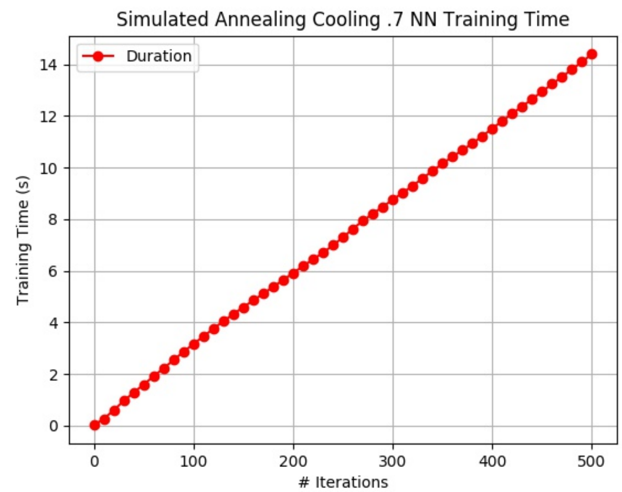
Permanent Visa Applicant NN Learning Curve



Permanent Visa Applicant NN Learning Curve



Permanent Visa Applicant NN Learning Curve



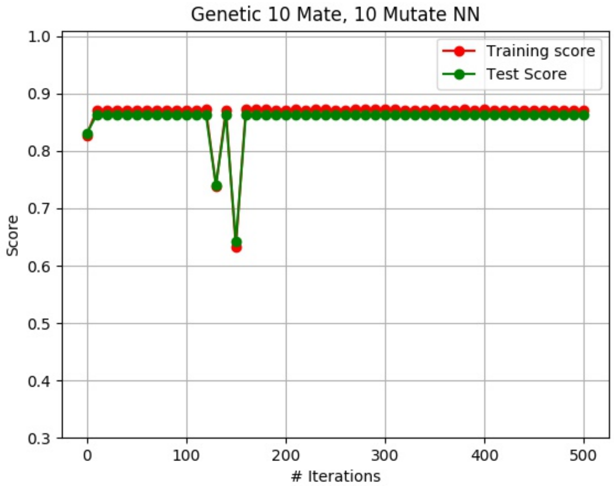
Permanent Visa Applicant NN Learning Curve

4) Genetic Algorithms

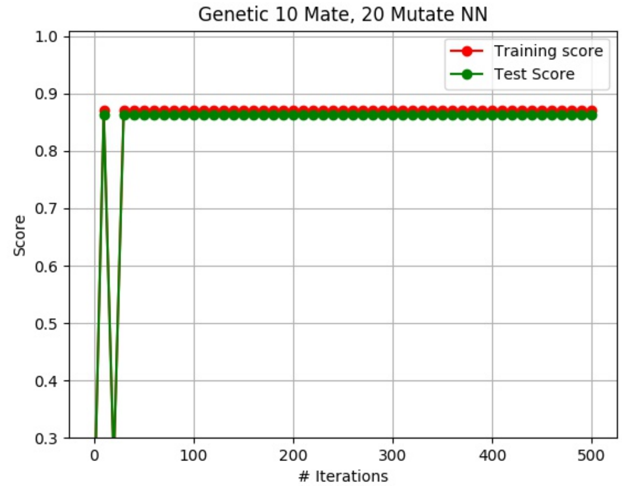
Overview

The fourth weight-finding algorithm used was a genetic algorithm. Genetic algorithms works by starting with an initial solution and then making modifications in an attempt to improve the solution. The modifications generally allowed are mutation (changing random parts of the solution), crossover (taking specific sections from various solutions and combining them), and selection (selecting certain sections from a solution to use again). In the context of a neural network, we can use genetic algorithms to make modifications to our network's weights.

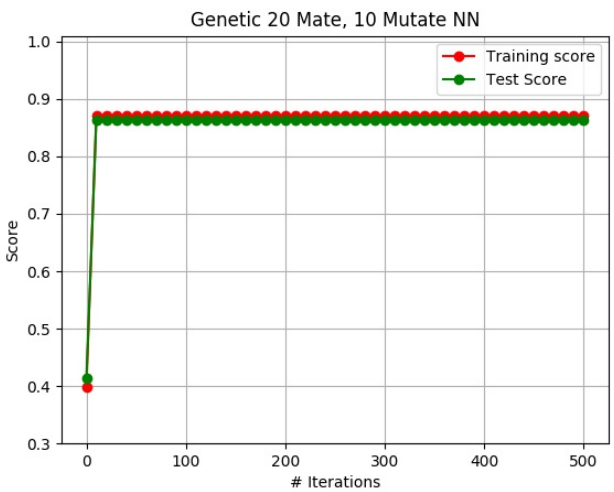
In our testing, a population size of 50 was used in order to get a diverse initial sampling of possible solutions. The number of instances to mate (aka crossover) and to mutate was varied throughout the trials.



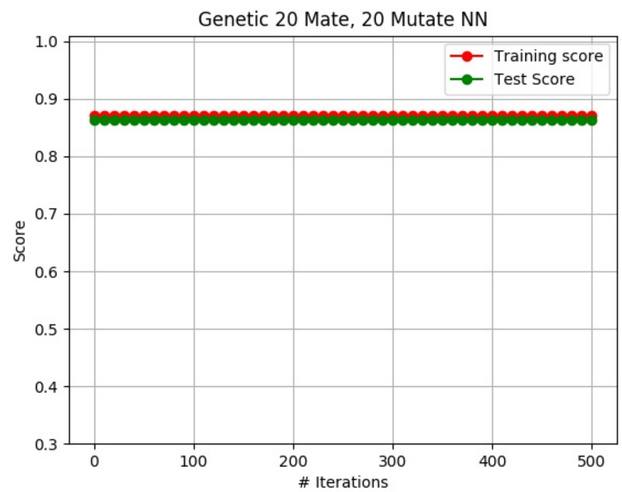
Permanent Visa Applicant NN Learning Curve



Permanent Visa Applicant NN Learning Curve



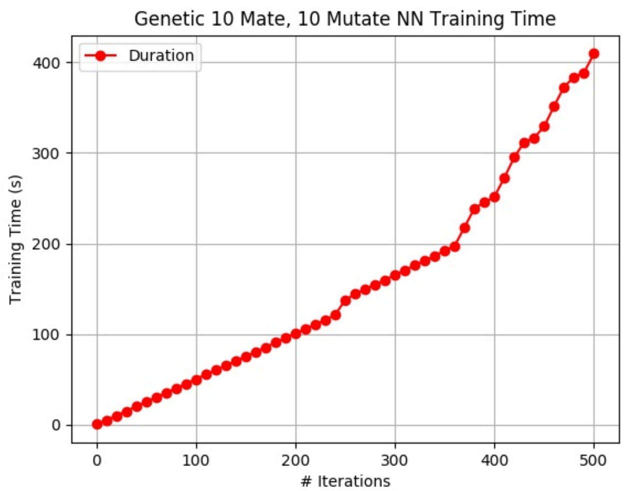
Permanent Visa Applicant NN Learning Curve



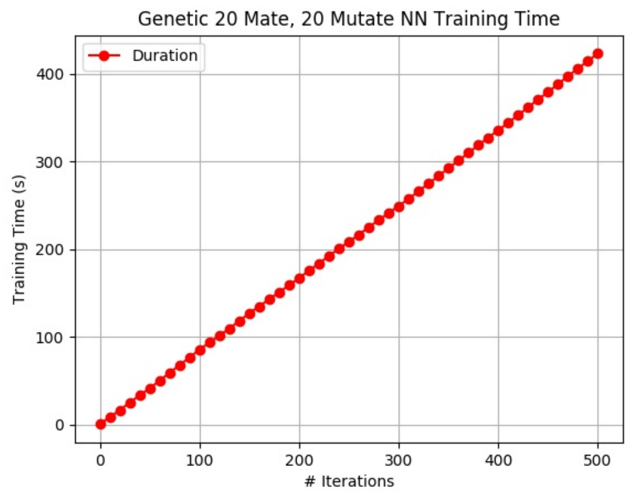
Permanent Visa Applicant NN Learning Curve

Trials were run with the following mate/mutate combinations: 10/10, 10/20, /20/10, and 20/20. All trials were run with a population size of 50. As can be observed from the graphs, both mating and mutating appeared to be highly effective in finding the optimal solution. In fact, the optimal solution converged quite quickly–in under 30 iterations for each trial. This is partly due to the fact that the data is easily learnable and does not appear to maintain very many trapping local maximums.

Training times scaled roughly linearly with the number of iterations ran. This makese sense because, from a performance perspective, the number of mutation calculations from iteration to iteration is more or less the same.



Permanent Visa Applicant NN Learning Curve



Permanent Visa Applicant NN Learning Curve

Conclusion

The optimization algorithms used all inevitably produced similar results. What differed, however, was the amount of training time required, the tuning of parameters, and how many iterations were required to achieve a consisent, optimal solution. All-in-all, the genetic algorithm approach consistantly produced the best results for our network. This makes sense, as the data was rather homogenous and there appeared to be very few outliers. By learning the training data well, the learner was able to perform similarly well on the (nearly identical) test data.

Part 2: Optimization Problem Domains

Introduction

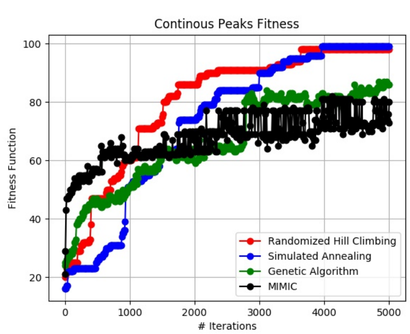
For this part of the assignment, three different optimization problems are examined: continous peaks, the traveling salesman, and flip flop. The three optimization techniques from above are used (randomized hill climbing, simulated annealing, and genetic algorithms) as well as another algorithm, MIMIC.

MIMIC, similar to the other algorithms, works to find the globally optimal solution. Unlike the other algorithms, however, it retains knowledge of previous iterations and uses this information to more efficiently find better solutions. MIMIC is particularly strong in regards to problems that maintain patterns between subsets of thier parameters.

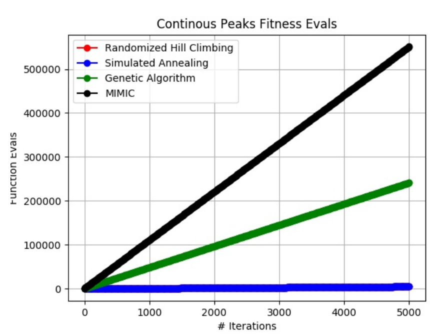
1) Continous Peaks

Overview

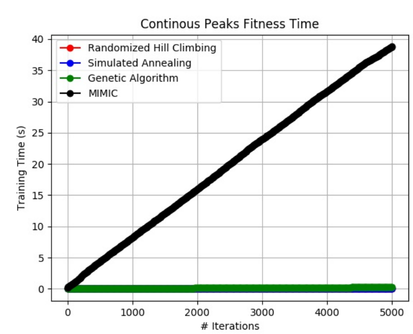
The continous peaks problem is an extension of the four peaks problem that allows for a wide variety of local maximums. This algorithm is particularly interesting due to the potentially large number of local maximums. It is especially difficult for an algorithm that cannot escape local peaks to perform well.



Permanent Visa Applicant NN Learning Curve



Permanent Visa Applicant NN Learning Curve



Permanent Visa Applicant NN Learning Curve

2) Traveling Salesman

Overview

The traveling salesman problem is a frequently used, classic problem that focuses on a salesperson trying to minimize their round-trip distance between any number of cities. This problem is particularly interesting due to it's real-world implications, such as route planing for delivery trucks and everyday errands–it also has no known polynomial time solution!

3) Flip flop

Overview

The flipflop problem is another common optimization, where one attempts to count the number of bits that alternate with its next neighbor in a bit string. This problem is particularly interesting because, since the strings are randomized, there is significant potential for a large number of local mimimum and maximums.

Conclusion

asdf