Wesley Smith

CS7641 – Assignment 2 – Fall 2018

# Introduction

This paper discusses four randomized optimization algorithms and their applications in both feedforward neural network training and several generic optimization problems. An *optimization problem* is defined as the process of maximizing a *fitness function* based on a set of inputs. It is common to consider only bit-string input spaces, however, neural network optimization considers vectors of continuous values.

The code used is borrowed with gratitude from Pushkar Kolhe’s ABAGAIL library [1] and Jonathan Tay’s ABAGAIL test harnesses custom-tailored for this assignment [2]. Parameters and datasets are modified accordingly.

## Overview of Algorithms

*Randomized Hill Climbing (RHC)* – A simple hill climbing algorithm which retains one maximally-fit parameter set at a time. RHC attempts to hill climb by comparing the best parameters to a neighbor, defined herein as having Hamming distance of 1. If the neighboring point’s fitness is higher, then this point is retained. Thus, RHC monotonically increases with increasing training iterations.

*Simulated Annealing (SA)* – Employs all the same techniques as RHC, except that points with lower fitness are probabilistically accepted over more fit alternatives. The likelihood of accepting such a decrease in fitness is determined by both the *temperature* – a hyperparameter supplied to the algorithm – and by the difference in fitness [3]. Higher temperatures result in higher likelihood of accepting lower-fitness parameters. A *cooling factor* is supplied to the algorithm, which multiplicatively decreases the temperature with each training iteration. The effect of the temperature and its decay is to have the algorithm initially explore the parameter space while eventually exploiting local maxima.

*Genetic Algorithms (GA)* – GAs are defined based on an elaborate analogy to biological evolution. GAs differ from hill climbing algorithms in that they maintain a *population* of parameter sets. At each training iteration, all *individuals* in the population are trained on the data and weighted based on their relative fitness versus the fitness of the population as a whole. A set number of individuals are selected for *crossover* probabilistically based on these weights – more fit individuals having higher likelihood of selection. The crossover operation can be specified in different ways, but it is always a way of breeding two individuals to produce an offspring (or two). The remainder of the population is sampled according to the same probabilities. Finally, a set number of individuals are *mutated* to allow for exploration of neighboring points. Through the crossover and mutation operators, as well as by maintaining a probabilistically-chosen population, GAs have a very robust ability to explore large parameter spaces.

*MIMIC* – MIMIC is the only algorithm discussed herein which tries to sample candidate parameter sets in a statistically meaningful way. Initially introduced in [4], the algorithm used in ABAGAIL uses the dependency-tree extension detailed in [5]. For each training iteration, MIMIC retains the top performers, and uses them to estimate the probability density across the parameter values. Pairwise conditional probabilities are used extensively, and provide the basis for the dependency trees which are used to sample future candidate hypothesis. MIMIC is therefore highly performant in problems for which the input parameters are highly inter-related.

# Part 1 - Neural Network Training

## Dataset / Network Recap

The focus of Part 1 is to compare randomized optimization techniques to backpropagation for neural network training as performed in Assignment 1 (A1). I chose the Adult dataset for a comparative analysis. In A1, the neural network that performed best had two hidden layers consisting of 100 nodes each. The dataset, after one-hot encoding, had 104 features. Combining this input size along with a single output node results in a network with 20604 links, each having their own weight. These weights represent the dimension of the random search space, i.e. the algorithms randomly search through 20604-length vectors.

When trained using backpropagation on A1, this network achieved a train, test, and validation balanced accuracy[[1]](#footnote-1) of 86%, 78%, and 77%, respectively.

## Randomized Search – Experimental Methodology

The same preprocessing steps were applied as in A1, namely dropping entries with missing data, one-hot encoding of categorical features, and min-max scaling. Of 30,162 samples, 30% of the data was split into a holdout test set. The remainder was split into 80% training data and 20% validation data.

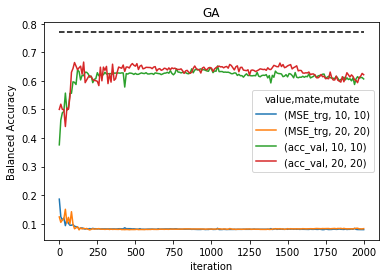
As noted above, RO algorithms are commonly compared based on benchmark optimization problems with solutions consisting of bit-strings. NN weights are continuous quantities. Consider a solution vector of 20604 bits; this vector has potential solutions. Converting each of these bits to a continuous quantity results in an infinite solution space of enormous dimensionality. Note that all weights are constrained to the interval [-0.5, 0.5], i.e., initializing consists of sampling all weights along that interval, while picking a neighbor consists of re-sampling a random neighbor along the same interval.

NN weights were trained using RHC, SA, and GA. For all algorithms, a fixed number of training iterations were used: 5000 for RHC and SA, 2000 for GA due to GA’s significantly longer execution time per training iteration. The fitness function to maximize was the inverse of the mean square error on the training data. In A1, the supervised learning objective was to maximize performance on validation data using K-fold cross-validation and validation/learning curve analysis. RO algorithms did not afford such a robust analysis.

## Randomized Hill Climbing

The shortcomings using RHC for a continuous search space this large are apparent. Over 5000 iterations, even if the algorithm was lucky enough to pick 5000 unique weights to modify, it would still have only chosen less than 25% of the available weights to modify, meaning that at least 75% of the weights remain initial random guesses.

# /var/folders/br/_5lsfr9n2jgdnl7v5pnry8j40000gn/T/com.microsoft.Word/Content.MSO/4CA3E9A9.tmp/var/folders/br/_5lsfr9n2jgdnl7v5pnry8j40000gn/T/com.microsoft.Word/Content.MSO/CC3F629F.tmp



**Figure 1.** Neural network learning curves for RO algorithms and backpropagation. A line at 77% shows the validation accuracy achieved in A1.

**Table 1.** Accuracy and timing comparison for neural network learning. Accuracy scores for A2 are the maximum values observed across all iterations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Validation Accuracy** | **Test Accuracy** | **Train time (sec) per iteration** |
| RHC | 72.1% | 72.3% | 1.88 |
| SA | 73.6% | 73.2% | 1.62 |
| GA | 67.3% | 68.1% | 56.7 |
| Backprop (A1) | 78.0% | 77.1% | 0.011 |
| Backprop (A2) | 75.4% | 74.9% | 8.8 |

# Part 2 – Optimization Problems

## Continuous Peaks – GA

The Continuous Peaks (CP) problem[[2]](#footnote-2) is an extension of the Four Peaks problem[[3]](#footnote-3), in which the fitness function depends on the size of contiguous strings of 0s and 1s. For CP, an input vector X of N bits as well as an integer parameter T are supplied. The fitness function is calculated as the sum of the length of the largest string of consecutive ones and that of the largest string of consecutive zeros. An additional reward of N is added if both Max(0) and Max(1) are greater than T. This creates the presence of a global optimum that is reachable by algorithms such as GA but hard to find for algorithms such as RHC.

* Most interesting T: 15
* Easy to show random hill climbing gets stuck in local optima
* SA is often capable of finding the reward
* GA performs very well
* MIMIC does ok – seems to have good variance, but when it fails, it is pretty bad

## Traveling Salesman – GA?

The traveling salesman problem (TSP) is the classic NP-hard problem of finding, between a number of nodes whose connecting distances are known, the shortest overall route which traverses all nodes and ends at the same node at which it started.

How is this modelled in a binary setting?

Initial impressions:

* RHC and SA are comparably not great
* GA has high highs and low lows
* MIMIC does very poorly again
  + Very low lows, low highs, large variance

## FlipFlop – SA/MIMIC

“FlipFlop” (FF) is fun name for a made-up optimization problem where maximum fitness is achieved by having an alternating bit pattern across the entire input string. The fitness function is simple: it is the sum of bits whose next bit is its complement. For an input of length N, the maximum fitness value is N-1.

Initial impressions:

* MIMIC finds near optimum performance very quickly
* GA has a lot of variance, relatively low numbers
* SA actually finds high performance at max iterations
* RHC is RHC

## Knapsack – MIMIC

Another classic NP-hard combinatorial optimization problem. The problem defines N items each with an individual weight and value. The fitness function to maximize is the sum of the product of all individual weights and values, such that the total weight is less than sum threshold W. The knapsack analogy is drawn from the problem of a hiker or soldier trying to maximize the quality of the load that they can carry.

I set N=40, the initial weights and values to sum number between 0 and 50, and the maximum weight to be 600 (30% of the maximum weight, i.e. if all item weights were 50). I used the 0-1 knapsack problem variant, meaning that items are either included once or not at all, making the input vector a bit-string.

# References

1. <https://github.com/pushkar/ABAGAIL>
2. <https://github.com/JonathanTay/CS-7641-assignment-2>
3. <https://link-springer-com.prx.library.gatech.edu/content/pdf/10.1007/s10479-005-3971-7.pdf>
4. <https://www.cc.gatech.edu/~isbell/papers/isbell-mimic-nips-1997.pdf>
5. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.8826&rep=rep1&type=pdf>

1. Balanced accuracy was used because the dataset contained approximately 75% negative samples [↑](#footnote-ref-1)
2. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.8826&rep=rep1&type=pdf [↑](#footnote-ref-2)
3. https://pdfs.semanticscholar.org/cd4f/e89d8dd6060e2957041f90fc699a30058d01.pdf [↑](#footnote-ref-3)