A Comparative Analysis of Randomized Optimization Algorithms

Michael Walton
Department of Computer Science, Georgia Institute of Technology

October 19, 2015

Abstract

In this work we compare the dynamics and performance of four randomized optimization algorithms. The capabilities and limitations of random hill climbing (RHC), simulated annealing (SA), genetic algorithms (GA) and Mutual Information Maximization for Input Clustering (MIMIC) are explored. Local optimization methods are applied to finding optimal weights for neural network in with respect to a digit recognition task. All four algorithms are then applied to three well known optimization problems. We conclude with general analysis comparing the strengths and weaknesses of these methods.

1 Introduction

Optimization is a diverse research domain in which a plethora of different algorithms have been explored to resolve problems with unique complexities and applications. The area with which this paper is primarily concerned is local random search methods; these algorithms tend to be targeted at problems characterized by non-convex optimization surfaces in which there may be many local minima with unknown distribution in the space. This difficulty leads to the possibility that local search methods will discover a local optimum, but may get 'stuck' before reaching a global optimum (such as a naive implementation of hill climbing in which the area around the current optimum is sampled and steps are taken in the direction of maximal increase until a convergence criteria). Different methods handle this issue in different ways, but generally the high-level concept is to use iterative stochastic sampling to cover a larger volume of the parameter space.

The randomized optimization algorithms considered in the present study are random hill climbing (RHC), simulated annealing (SA), genetic algorithms (GA) and Mutual Information Maximization for Input Clustering (MIMIC). Implementations are based on the ABAGAIL library [5] The python modules `scipy,` `numpy,` and `scikit-learn` [3] [4] are used extensively throughout the implementation.

2 Methods

In the sections that follow, we will provide a brief overview of the digit classification dataset and the three optimization problems used in this study. In addition, we will briefly discuss the implementation details for the considered algorithms.

2.1 Digit classification Dataset

The first dataset considered in this study is a collection of 8 x 8p gray-scale images of handwritten digits. Each input instance $x \in X$ is projected, or 'flattened' from $x \in \mathbb{R}^{8\times 8}$ into a 64-dimensional vector $\hat{x} \in \mathbb{R}^{64}$ the new feature array $\hat{X} \in \mathbb{R}^{64\times m}$ where $m = 1796$ is the number of instances in the dataset; target labels valued $\{0, 1...9\}$ are encoded as a vector of binary strings $\boldsymbol{y}$ where, given a target label n the n^{th} bit of the corresponding string is 1 and 0 for all other bit positions. Our objective is to derive a hypothesis $h \in H, h : x \in \mathbb{R}^{64} \mapsto y \in \{0, 1...9\}$ approximating the target concept $c(x)$. We define the preferred hypothesis as the h which minimizes $error_{\mathbb{D}}(h)$ on the actual distribution $\mathbb{D}$ by learning on the training set $S \subset D$ where D is the available dataset of (instance, label) pairs $\{(x_i, y_i)\}$ The objective is some tbd. 'best' function which partitions the instance space by mapping each example $x \in X$ to a corresponding bitstring $y \in \boldsymbol{y}$ where the n^{th} bit of y indicates the target label index.

In the previous study we used two Parkinson's disease dysphonia datasets to illustrate the capabilities and limitations of several supervised learning algorithms. While this was demonstrative of a possible real-world application of these methods on a non-trivial dataset, the study suffered from a lack of understanding of the problem domain and input feature space. We have elected to use the digit classification problem as an alternative because of its straightforward specification and extensive documentation. It is our hope that this will afford more opportunity to highlight the algorithms and significance of the data with respect to machine learning rather than placing the emphasis on the complexity of the problem domain.

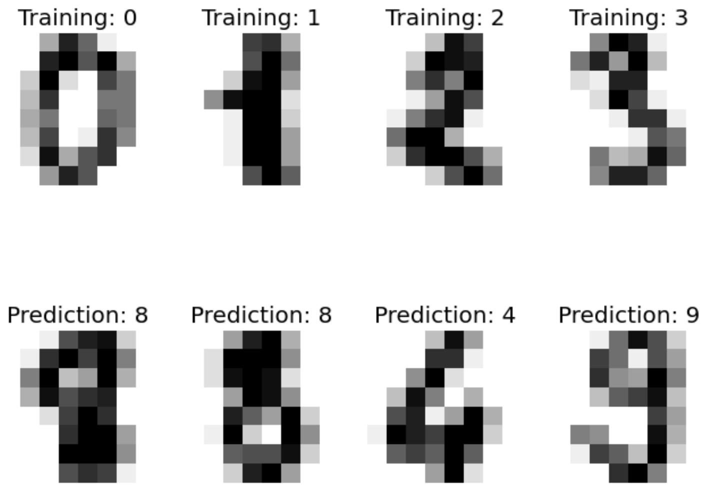


Figure 1: Hand Written Digits Example [4]

2.2 Optimization Problems

The four peaks problem is predefined in ABAGAIL as `FourPeaksEvaluationFunction` and is adapted from [1]. The problem is defined in terms of bitstrings of length 100 where a parameter T defines a threshold for a reward term of the fitness function. The goal is to maximize $z(x)$, the longest run of zeros starting from position 100 and moving backwards in the bitstring and $o(x)$ the longest run of 1s starting from position 1 in the bitstring. The reward term increments the fitness value by some large number if both objectives are exceed the threshold T, that is if $o(x) > T \wedge z(x) > T$. The problem has two local maxima and two global maxima, hence the name 'four peaks'.

The k-coloring optimization problem is widely studied in computer science, in our implementation we use the `MaxKColorFitnessFunction` from ABAGAIL. Given a graph G with k possible colors for each of its verticies $V \in G$, the k-coloring of G is a color assignment st. no two adjacent V have the same color. The n-queens problem is also widely explored in computer science and seeks to find a placement of n queens in a standard chess board st. no queen may capture another in a single turn. We implement this scenario in ABAGAIL using `NQueensFitnessFunction` and `NQueensBoardGame`. The traveling salesman problem is yet another extensively explored and well defined optimization problem, our experiments use ABAGAIL's `TravelingSalesmanRouteEvaluationFunction`. In the traveling salesman problem, we consider the traversal of a graph G where verticies and edges $G =$

(V, E) are thought of as cities and distances respectively. The objective is to find a traversal st. every city is visited, the initial and destination cities are the same vertex, and the length of the traversal path is minimized.

2.3 Algorithms and Evaluation

Here we will discuss the application of RHC, SA and GA to the neural network optimization problem. In the case of the other optimization tasks, the reader should consider the general fitness function of the corresponding problem f however in the context of the digit classifier ANN, we discuss the algorithms in the context of minimizing the sum squared error E. The three optimization algorithms that were applied to finding weights for a neural network operated on a three layer model with 20 hidden units, 64 input units, 10 output units (corresponding to the dimensionality of the image instances and output labels respectively). Simulated annealing was run with initial temperature parameter $T = 10^{11}$ cooling rate $c = .95$. The genetic algorithm was run with the `StandardGeneticAlgorithm` ABAGAIL class with a population size $p = 200$ mating rate $m = 100$ which is the number of individuals which mate per iteration and mutation rate $\mu = 10$ which gives the number of individuals which mutate per iteration.

In randomized hill climbing, the weights of the network are initialized to a set of small random values. It is helpful to think of the network as having a particular state on each iteration which is a vector of weights ω in the high dimensional weight space. The network computes the local gradient of the error with respect to the weight vector $\frac{\partial E}{\partial \omega}$ and updates ω along the path of steepest descent (in the case of error minimization). This can be thought of as a greedy search through the weight space of the network. Simulated annealing conducts a randomized version of this search in a similar way, however in the case where on an iteration $t, \neg\exists \omega_{t+1}$ st. $E_{t+1} < E_t$ an ω_{t+1} local to ω_t is selected with probability proportional to the temperature term T. On each iteration, $T \leftarrow T - c$ where c is the cooling rate of the algorithm.

Genetic algorithms conduct a similar stochastic search through the parameter space of ω (and equivalently the hypothesis space H of possible neural networks given the constrained topology). However the implementation and representational structure differ from SA and RHC. The GA represents unique weight vectors ω as individuals in a randomly generated population Ω. On each iteration, the fitness (in this context taken to be the sum of squared error) of each individual $E(\omega)$ is evaluated and the m individuals with the least error are selected for breeding via a crossover procedure on the bits corresponding to each weight parameter in ω. μ of the individuals in the new population are mutated by bit-flipping a single weight according to ABAGAIL's `DiscreteChangeOneMutation`.

The Mutual Information Maximization for Input Clustering (MIMIC) [2] algorithm is applied along with the aforementioned three to the other optimization problems. MIMIC estimates probability densities over the input space and iteratively samples from the top N-percentile of the estimate to optimize the fitness measure. On each iteration, MIMIC updates its estimation p^{θ} and decrements $\theta \leftarrow \theta - \epsilon$ st. an increasing smaller region of the parameter space is sampled. Good estimates $\hat{p}_{\pi}$ of the true probability distribution P minimize the Kullback-Leibler divergence $D_{KL}(P||\hat{p}_{\pi})$. In the case where the true distribution P is not known, we may estimate P using a dependency tree which maximizes the mutual information between each feature node and its parent. Therefore, on a particular iteration t, a best estimate of the θ^{th} percentile of the data $P^{\theta}_{t+1}(x)$ is given by the maximum spanning dependency tree with respect to the mutual information I over the input features of the data.

In the neural network digit classification optimization problem, the 'fitness' measure is defined in ABAGAIL as `SumOfSquaresError`. In the context of our neural network, given training data $D, |D| = n$, the fitness function is simply the mean squared error of the network output. Therefore the full optimization problem is given by:

$$\underset{h \in H}{\operatorname{argmin}} \left[\frac{1}{n} \sum_{i=1}^{n} (\hat{Y}_i^h - Y_i)^2 \right] \tag{1}$$

Definition for the four peaks, k-colors, n-queens and traveling salesman problems are given in the Optimization Problems sub-section of this paper.

3 Results

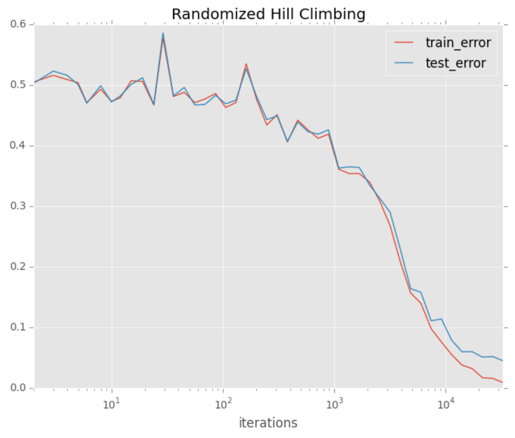
3.1 Model Evaluation

3.2 Performance Comparison

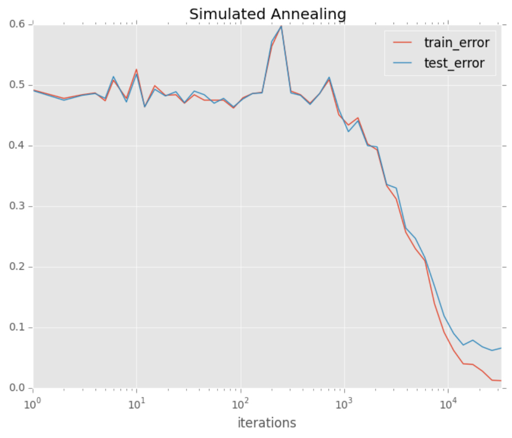
4 Conclusion

References

[1] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. Technical report, Pittsburgh, PA, USA, 1995.



(a) Randomized Hill Climbing



(b) Simulated Annealing

Figure 2: Neural Network Optimization Algorithm Comparison

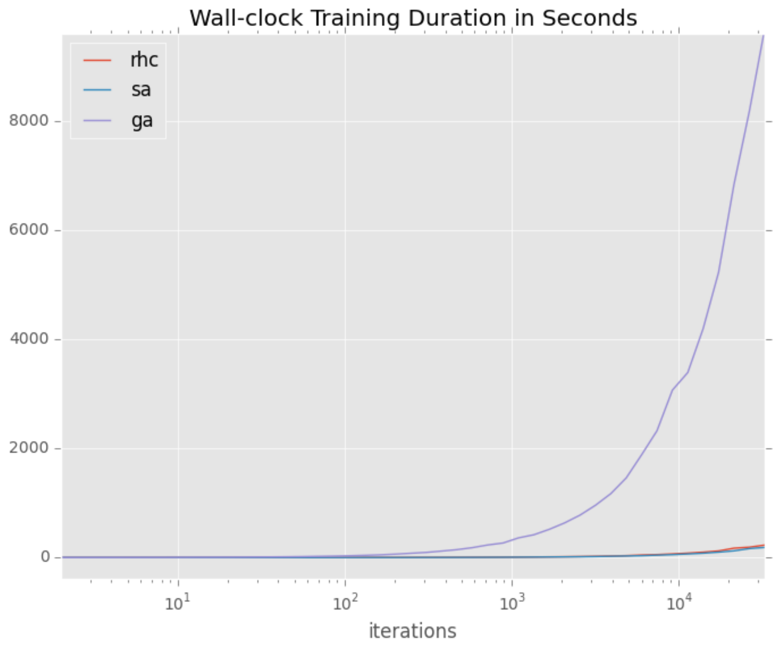
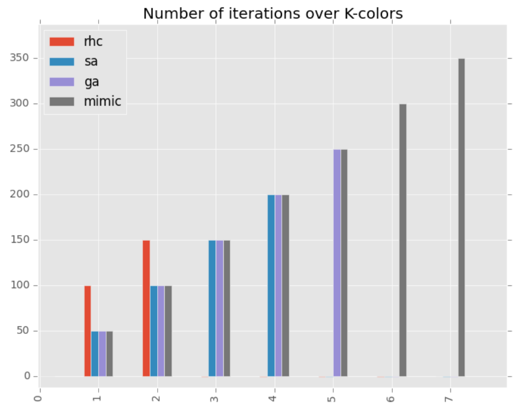
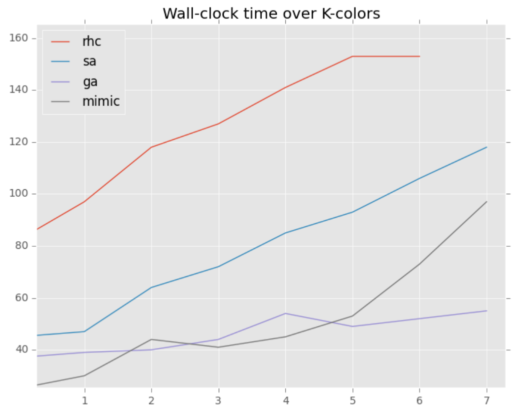


Figure 3: Wall-clock Algorithm Runtimes



(a) Iterations to Find k-coloring



(b) Wall-clock Runtime in MS

Figure 4: K-Colors Algorithm Comparison

[2] Jeremy S. De Bonet, Charles L. Isbell, Jr., and Paul Viola. Mimic: Finding optima by estimating probability densities. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, page 424. The MIT Press, 1996.

[3] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2015-09-15].

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Pushkar. Abagail, 2015.