1 Introduction

1.1 Experiments

This project uses three algorithms: simulated annealing, random hill-climbing and genetic algorithms – on different kinds of problems.

The first problem is training neural network weights on the Mushrooms dataset from Kaggle. Instead of using a standard backprop algorithm like gradient descent, this project investigates the use of the three algorithms for adjusting the weights of a neural network.

The other two problems are more of a pure algorithmic flavor: they are four peaks and k-color.

Four peaks is defined as the following (taken from [1]):

$$f(\vec{x}, T) = \max\left[tail(0, \vec{x}), head(1, \vec{x})\right] + R(\vec{x}, T)$$

where

$tail(b, \vec{x})$ = number of trailing b's in $\vec{x}$

$head(b, \vec{x})$ = number of leading b's in $\vec{x}$

$$R(\vec{x}, t) = \begin{cases} N & tail(0, \vec{x}) > Tandhead(1, \vec{x}) > T \\ 0 & otherwise \end{cases}$$

We want to maximize this number for a given threshold T.

K-color is a problem that attempts to find a coloring for a graph $G(V, E)$ such that the number of edges that have two vertices of the same color is minimized. K-color attempts to minimize the evaluation function

$$\sum_{\{u,v\} \in E} c_{uv}$$

where for each $\{u, v\}$ in all the edges, $c_{uv} = 1$ if u and v are of the same color and zero otherwise.

2 The algorithms

2.1 Simulated annealing

The simulated annealing algorithms used in this project were taken from the library simanneal. The simulated annealing algorithm uses a move() function (to find a local neighbor) and an energy() function (to evaluate the neighbor) and picks a neighbor if it has a higher energy function or otherwise picks a neighbor with some probability proportional to the current temperature, which decreases after every iteration, and also proportional to the deviation from the current energy. The default parameters in this library are: Tmax: 25000.0, Tmin: 2.5, steps: 50000, updates: 100

2.2 Randomized hill-climbing

The hill climbing was manually implemented. It was implemented with outer iterations and inner iterations, such that for each outer iteration, a random location in the space was chosen. Then, in each inner iteration, several local candidate neigbors were found, and the neighbor with the highest evaluation function (if higher than the current evaluation function) was chosen. The best point over all outer iterations was chosen.

2.3 Genetic algorithm

The genetic algorithms used to solve these problems were taken from scipy.optimize.differentialevolution. This algorithm selects everal local neighbors and selects the popsize * len(x) best at each iteration (where x is the parameter to the fitness function). Between iteration, crossover occurs to generate new points.

3 Results

4 Analysis

5 Sources

[1] https://www.cc.gatech.edu/ isbell/tutorials/mimic-tutorial.pdf

[2] https://www.kaggle.com/uciml/mushroom-classification