**CS-7641 Machine Learning:**

**Randomized Optimization**

Junle Lu

[junle.lu@gatech.edu](mailto:junle.lu@gatech.edu)

Georgia Institute of Technology

October 14, 2018

Abstract – In this paper, we investigated and analyzed four random search algorithms: Randomized Hill Climbing (RHB), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC. The experiments and analysis consist two parts: neural network optimizer and optimization problems. The first three algorithms are utilized as weight optimizer for neural network to find the optimal weight for a given dataset from the previous assignment. The performance is evaluated and analyzed. To better understand the random search algorithms, three optimization problems are chosen to demonstrate the advantages and disadvantages of each algorithm: Travel Salesman Problem (TSP), Flip Flop (FF), Max K-coloring.

1. Random Search Optimizer for Neural Network

1.1 Introduction

In the last paper, the performance of a neural network was analyzed with two datasets, but only one is chosen to be used for the experiments: gender recognition by voice.

**Gender Recognition by Voice**: this database was created to identify a voice as male or female, based upon acoustic properties of the voice and speech. The dataset consists of 3,168 recorded voice samples, collected from male and female speakers. The voice samples are pre-processed by acoustic analysis in R using the seewave and tuneR packages, with an analyzed frequency range of 0hz-280hz. It has 3168 instance and 20 features.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Weight optimizer** | **Hidden layers** | **Neuron** | **Train score** | **Test score** | **CV score** | **Iteration** | **Runtime** |
| lbfgs | 2 | 17 | 0.934 | 0.934 | 0.931 | 2286 | Slow |
| sgd | 4 | 17 | 0.625 | 0.614 | 0.613 | 20 | Fast |
| adam | 11 | 17 | 0.865 | 0.866 | 0.843 | 46 | Fast |

Table 1: Performance Summary for MLPClassifier (sk-learn)

Although the dataset was analyzed with Multi-layer Perceptron classifier (scikit-learn library), but the multi-layer neural network from ABAGAIL library is used for analysis with existing random search algorithms. The optimal values hyperparameters are determined from previous experiments (table 1) and will be used in this paper. Following hyperparameters are used to build the neural network implemented in ABAGIAL:

* Number of hidden layers: 2
* Number of nodes in a hidden layer: 17

1.2 Experiment Setup

Each optimization algorithm (RHC, GA, and SA) utilized to find the optimal weights for the neural network in multiple experiments. All algorithms are implemented in ABAGAIL package. In addition to each algorithm’s hyperparameters, the following variables are investigated and analyzed for all the algorithms against number of iterations: training score, testing score, training mean square error, testing mean square error and run-time. Although cross-validation is excellent to determine if the training model is overfitting or underfitting, but it is not available in ABAGIAL package. However, the performance of the algorithms can still be compared and analyzed. The dataset is split into training (70%) and testing (30%) datasets.

Each algorithm is trained for a max number of iterations, and the data is collected at every 10 iterations to reduce the program run-time. The neural network is trained on each iteration. For every 10 iterations, the optimal weights are applied to the neural network to collect the mentioned data. Then, the neural network is restored with its sub-optimal weights to continue its training.

1.3 Randomized Hill Climbing

Randomized Hill Climbing (RHC) is a simple and yet an effective randomized optimization algorithm in cases. It starts at a random point. For every iteration, it gauges its neighbors’ fitness. It will move to the point with the best fitness. It repeats this search until none of the neighbors have better fitness evaluation. At this point, algorithm either have reach the search goal or it is stuck at a local optimum location.

From the figure above, it has been until it convertge. The number of restart is the only hyperparameter for this algorithms, but the experiment did not show that its beneficial for this data set.

1.4 Simulated Annealing

Simulated annealing is a randomized optimization algorithm inspired by the concept of annealing in metallurgy, whereby a metal is heated up and cooled down in a controlled manner to allow the molecules in the metal to align, increasing the overall strength and quality of the metal. The algorithm takes this general idea and extends it to data, using several concepts from probability theory and statistics.

The general algorithm begins at a random point. It then begins its first iteration by sampling a random point from elsewhere in the data. The distance between these two points, as would typically be defined by some distance or neighbor function, is instead based off of a probability distribution, such as the Boltzmann distribution, often used to model energy distributions of large systems of particles. This distance is then considered towards some goal, such as increasing our fitness function or lowering a cost function. The algorithm then decides whether or not to go to this point, depending on whether it moves us toward the goal. However, since data distributions are often filled with local optima that are some distance away from the goal, the algorithm may decide to move to a point farther from the goal in order to avoid becoming stuck in one of these local optima, conferring a substantial advantage to the algorithm over randomized hill climbing, which is prone to getting stuck in such regions; at least in theory.

True to the theme of heating up metal and cooling it in a controlled manner, there are two hyperparameters of concern when it comes to simulated annealing: cooling rate (CR), and starting temperature (Temp). We begin with an analysis of cooling rate on the algorithm. Cooling rate is a constant, anywhere from 0 to 1, defining how quickly the temperature is decreasing with each iteration (lower values cool faster). This decrease in temperature corresponds to a decrease in how much the algorithm can jump between points; that is to say, the lower the temperature at any iteration, the more the algorithm is inclined to move towards points that move us closer toward the goal, increasing the chances of getting stuck in a local optima. The general idea of the algorithm then, is that we sample many random points in the beginning, in an attempt to determine a best optimum to move towards, slowly accelerating towards this optimum, and localizing the space of available points as the algorithm continues. But what value is optimal for cooling rate? We picked 6 different values and tried them on our dataset