Open Lab 2

Solving Problems By Local Search

CSCI 4350 - Introduction to Artificial Intelligence

Due: Oct. 18 @ 11:00pm

Overview

Develop a software agent in Python to find the maximum value of the Sum of Gaussians (SoG) function.

Procedure

1. Create a Python program which uses greedy local search (gradient ascent) to obtain the maximum value of the SoG function, $G()$, in d dimensions (`greedy.py`):
   - The program should take 3 command-line arguments: (**integer**: random number seed (s), **integer**: number of dimensions (d) for the SoG function, **integer**: number of Gaussians (n) for the SoG function).
   - The program should start in a random location $\boldsymbol{x}$ in the [0,10] d-cube, where $\boldsymbol{x}$ is a d-dimensional vector.
   - The program should use a step size of ($0.01 * \nabla_{\boldsymbol{x}} G$) to perform gradient ascent where $\nabla_{\boldsymbol{x}} G = [\frac{\delta G}{\delta x_0}, \frac{\delta G}{\delta x_1}, \cdots, \frac{\delta G}{\delta x_{d-1}}]$.
   - The program should terminate when the value of the function no longer increases (within 1e-8 tolerance) **OR** at a maximum of 100000 iterations.
   - The program should print the location ($\boldsymbol{x}$) and SoG function value ($G(\boldsymbol{x})$) at each step (see requirements).
2. Create a Python program which uses simulated annealing (SA) to obtain the maximum value of the SoG function in D dimensions (`sa.py`):
   - The program should take 3 command-line arguments: (**integer**: random number seed (s), **integer**: number of dimensions (d) for the SoG function, **integer**: number of Gaussians (n) for the SoG function).
   - The program should start in a random location $\boldsymbol{x}$ in the $[0, 10]$ d-cube, where $\boldsymbol{x}$ is a d-dimensional vector.
   - The program should create an *annealing schedule* for the termperature (t), and slowly lowering t over time.
   - On each iteration, the program should generate a new location $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is a d-dimensional vector of random, uniform values in the range $[-0.05, 0.05]$, and choose to accept it or reject it based on the metropolis criterion:
     - if $G(\boldsymbol{y}) > G(\boldsymbol{x})$ then accept the move to $\boldsymbol{y}$
     - else accept the move to $\boldsymbol{y}$ with probability $e^{\left(\frac{G(\boldsymbol{y})-G(\boldsymbol{x})}{t}\right)}$

- The program should terminate at (a maximum of) 100000 iterations
- The program should print the location ($\boldsymbol{x}$) and SoG function value ($G(\boldsymbol{x})$) at each step (see requirements).

3. Utilize your programs to analyze the performance of the algorithms:
   - Use your greedy program to solve the SoG function for all combinations of d=1,2,3,5 and n =5,10,50,100.
   - Use your SA program to solve the SoG function for all combinations of d=1,2,3,5 and n =5,10,50,100.
     - Use 100 **unique, but corresponding** seed values for each case to ensure that you are solving the same problem using both algorithms.
   - Calculate the number of times that your SA program **out-performed and/or tied** your greedy program for each condition (within 1e-8 tolerance).
4. Write a report (at least 2 pages, single spaced, 12 point font, 1 inch margins, no more than four pages) describing:
   - the SoG function,
   - the code you developled to optimize the function,
   - the annealing schedule you settled on,
   - the performance of the code under various conditions (using the statistics above for justification),
   - any limitations of the overall approach,
   - and describe any additional implementation details that improved the performance of your code.

Requirements

- You must utilize the `uniform()` function (from numpy) for generating random numbers:

```
import numpy as np
rng = np.random.default_rng(seed)
vals = rng.uniform(size=d)
```

- You must utilize the SumofGaussians class to set up the problem (download: OLA2-support.zip).
- Use insightful comments in the code to illustrate what is happening on each step.
- Include a header in the source code and report with relevant assignment information.
- Your code should **only** print the current location, $\boldsymbol{x}$, and the value of the SoG function, $G(\boldsymbol{x})$, on each iteration:
  - Example, 1-D output:

```
7.15189366 0.06278163
7.14980459 0.06321930
7.14770360 0.06366198
7.14559057 0.06410975
7.14346539 0.06456269
7.14132794 0.06502088
...
```

```
* Example, 2-D output:
```

```
5.82914243 6.07541906 0.27940760
5.82723683 6.08143456 0.28340659
5.82531476 6.08750207 0.28747493
5.82337614 6.09362178 0.29161350
5.82142092 6.09979391 0.29582318
5.81944905 6.10601862 0.30010479
...
```

- Write your report such that a peer NOT taking this course would understand the problem, your approach to solving it, justification of various choices (annealing schedule, gradient step size, etc.), and your final comments.
- When running the code to gather statistics, you might prefer to print output for only the **final** iteration to speed up the data-gathering process (be sure to change it back to print for every iteration before submission though).
- Include a table of **all** of the **statistics** compiled for your report (not the raw data).
- Include at least one figure to illustrate the SoG function.
- All sources must be properly cited; failure to do so may result in accusations of plagiarism.
- Your report should be submitted in PDF format.

Submission

- A zipped file (.zip) containing (with **exact** filenames):
  - `greedy.py`
  - `sa.py`
  - `SumofGaussians.py`
  - `report.pdf`
- Typical command to zip your lab: `zip OLA2.zip greedy.py sa.py SumofGaussians.py report.pdf`
- Download your zip file and then use your PipelineMT credentials to log in and submit your zip file to the Open_Lab_2 dropbox: https://jupyterhub.cs.mtsu.edu/azuread/services/csci4350-assignments/