CMPSCI 383 Homework 3

YOUR NAME HERE

Assigned: Mar 22 2018; Due: Apr 4 2018 @ 11:59 PM EST

Abstract

To submit this assignment, upload your my_gradient.py and my_csp.py files to the Gradescope programming assignment. When uploading just upload both python files and do not put them in a folder to upload. Do not change function definitions or return types unless otherwise specified. Your work for all parts of this assignment must be your own (do not collaborate with other students in any way when completing this assignment).

1 Gradient Optimization (50 pts)

In this section you will be implementing gradient descent and Newton's method to optimize a continuous function. The function you will be optimizing is the Rosenbrock function:

$$f(x) = (a - x_0)^2 + b(x_1 - x_0^2)^2$$

In this function f maps a vector $x \in \mathbb{R}^2$ to a scalar in $\mathbb{R}$. There are N-dimensional extensions of the Rosenbrock function, but for this assignment you should only consider the 2-dimensional case. More about the Rosenbrock function can be read on Wikipedia `https://en.wikipedia.org/wiki/Rosenbrock_function`.

For this part of the assignment you must implement the following:

1. Rosenbrock function: Implement the 2-dimensional Rosenbrock function given above using the prototype function given in the *my_gradient.py* file. The parameters a and b should not be hard-coded as your implementation will be evlauted with different values of a and b.

2. Rosenbrock derivative: Complete the *rosenbrock_grad* function in the *my_gradient.py* file. This function should compute the derivative of the Rosenbrock function at a given vector x. The parameters a and b should not be hard-coded as we will evaluate your implementation with different values of a and b.

3. Rosebrock Hessian: Finish the *rosenbrock_hessian* function to compute the hessian of the Rosenbrock function at a given vector x. As with the previous parts a and b should not be hard-coded

4. Gradient Descent: Implement a gradient descent optimization function that follows the negative gradient of a function until convergence or for a set number of steps. The function *gradient_descent* takes as input a function f that evaluates to a scalar, a function that computes the gradient of f, an initial starting vector x_0, a step size parameter, a convergence threshold, and a maximum number of steps to be run.

   Recall that the main gradient descent step of a function f is:

   $$x_{k+1} = x_k - \alpha \nabla f(x)$$

   where α is the step size parameter and $\nabla f(x)$ is the derivative of f with respect to the vector x. Your implementation should step in the negative gradient direction until $|f(x_k) - f(x_{k-1})| < \epsilon$ where ϵ is the convergence threshold hold. This threshold should not be hard coded and you should use the threshold parameter in this check. Sometimes gradient descent will not converge in a reasonable amount of time so your implementation should stop the optimization procedure after a given number of steps.

5. Newton's method: You need to implement Newton's method for minimizing a function. This function should be similar to the gradient descent function, but make use of the Hessian.

   Recall that the minimization step of Newton's method is:

   $$x_{k+1} = x_k - \alpha \mathbf{H}^{-1} \nabla f(x)$$

   where $\mathbf{H}^{-1}$ is the Hessian of f at x.

There is additional code in the *my_gradient.py* file that will random generate start positions and then running gradient descent and Newton's method for different learning rates. You can use this to evaluate how fast each method converges and find the optimal step size. A hint for debugging your optimization function implementations you can create new simpler functions to optimize such as $f(x) = x^2 + 1$ and their derivatives.

To submit this part of the assignment just upload your *my_gradient.py* file to Gradescope. Your code will be evaluated on Gradescope for correctness of the Rosenbrock functions and optimization functions. The optimization functions will be test on functions that are not the Rosenbrock function so your implementation should be generic. You are not required to write up anything for this part of the assignment. However, as gradient optimization is a very common method to use in AI and machine learning algorithms you are in encouraged to play around with optimizing different functions.

2 Constraint Satisfaction Problems (50 pts)

In this part of the assignment you will be creating a constraint satisfaction solver for solving Sudoku puzzles. We have provided a *sudoku.py* file with methods for loading a Sudoku puzzle from a file and other methods helping you check your implementation. We also provided several puzzles in the 'puzzles' folder where harder puzzles have higher numbers. There is also a *my_csp.py* file with skeleton code that you will use to implement the backtracking search algorithm. For this part of the assignment you are required to implement the backtracking search algorithm described in Figure 6.5 of the textbook. Your algorithm should also use the minimum remaining value (MRV) heuristic. Your implementation will be evaluated by returning the correct solved puzzle and the number of guess for some held out puzzles.

There is some skeleton code provided in the *my_csp.py* file that you might find helpful for your implementation. You are not required to use these functions. The only required function is the solve method in the CSP_Solver class. You are not allowed to change the constructor of this class or the method signature or return type of the solve method. To submit your code for this part of the assignment upload the *my_csp.py* file to gradescope. To receive credit for both parts of the assignment you must upload both *my_csp.py* and *my_gradient.py* files at the same time and they can not be stored in a folder.