Project Report

Simon Zhan, EECS 127 05/02/2020

Booth function

In Booth function part, since this function's global minimum matches its local minimum, we could use any choice of gradient descent, momentum, or NAG to achieve minimum point. After contrasting between gd and NAG and gd and $Momentum$ under the same parameter, we find out that gd has a better performance, but all the algorithms can achieve the required error with similar path. (Contrasting graph can be seen in the Booth code part).
The parameters I set is as follow:

- $learning_rate = 0.02$
- $max_iteration = 2500$
- $alpha(nag_momentum) = 0.8$
- $tolerance = 1^{-8}$

Beale function

In the Beale function, both $Momentum$ and gd method can reach the global minimum point. However, the path for $Momentum$ and gd is different. (For each different path graph can be seen from the code). The parameters I set for gd algorithm are:

- $learning_rate = 1^{-4}$
- $max_iteration = 100000$

For the $Momentum$ approach, the parameters I set are as followed:

- $learning_rate = 1^{-4}$
- $max_iteration = 100000$
- $alpha = 0.85$
- $tolerance = 1^{-8}$

From the contrasting result and descent path, we can find out that gd method has more direct path and closer result to the global minimum point than $Momentum$ does (specific result can be seen from the coding block of Beale function).

Rosenbrock function

For the Rosenbrock function, I did not find out the optimal parameters for gd, $Momentum$, and NAG at the initial point we plan to start. Thus, I use $Adagrad$ method with parameters:

- $learning_rate = 2$
- $max_iteration = 500000000$
- $initial_point = (8, 9)$
- $tolerance = 1^{-10}$

However, during the exploration, I find out that if we change the initial point, there are some other methods can be implemented to achieve minimum. For example, if we change to coordinate $(4, -4.1)$, we actually can use gd to reach the minimum point.(Detail implementation can be seen from the code block).

Ackley function

Optimizing Ackley function is tricky, since it has lots of local minims and maxims, which might trap in those local area if the step-size is too small. Besides, if the step-size is too big, those SDG algorithm may also wandering around the surrounding region of the global minimum (can be seen from the function graph). Therefore, I implement a step size changing strategy in this case. At the beginning, I want the step-size to be big, since I don't want to trap into the local minimum. Then, I want the step size starts to decreasing to get into the convex part of the function visualized from the graph and to get more accurate result. Thus, I define my step-size as:

$$\begin{aligned} \eta_{\mathrm{t}} &= \mathrm{c}_1 / \left(\mathrm{t}^{\mathrm{a}} + \mathrm{c}_2\right) \\ \text{where} \quad & 0.5 < \mathrm{a} < 2 \\ & \mathrm{c1} > 0 \\ & \mathrm{c2} \geq 0 \end{aligned} \tag{1}$$

t is the number of iteration.
Then by using $Momentum$ method with the following parameters

- $c_1 = 10000$
- $a = 1$
- $c_2 = 15500$
- $learning_rate = calculated$
- $max_iteration = 1000000$
- $initial_point = (25, 20)$
- $tolerance = 1^{-5}$