Stochastic Gradient Descent and its application

Jungang Bu

March 2021

1 Introduction

Gradient Descent (GD) is one of the most popular optimization algorithms, and it is also the most commonly used algorithm to optimize neural networks. At the same time, each of the most advanced deep learning libraries includes the implementation of various variants of gradient descent algorithm. This article aims to provide readers with an intuitive understanding of the working principle of these algorithms and some applications. Let's first introduce the different variants of GD. Next, we will show the convergence rate of GD and Stochastic Gradient Descent (SGD). We will also show some examples of using GD and SGD, like finding the minimizer of different dimensional functions. Last but not least, we implement this technique into the finance field, like calibration for the volatility in the Black-Scholes Model.

2 Gradient Descent Variants

2.1 Batch Gradient Descent

Batch Gradient Descent (BGD), or Gradient Descent for simplicity, is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function $\mathbf{g(x)}$. In machine learning algorithm, sometimes it is necessary to build the loss function for the original model, and then optimize the loss function through the optimization algorithm in order to find the optimal parameters and minimize the value of the loss function. In the optimization algorithm of solving machine learning parameters, the gradient descent algorithm is widely used.

BGD is based one the observation that if the multi-variable function $g(\mathbf{x})$ is defined and differentiable in a neighborhood of a point $\mathbf{a}$, then $g(\mathbf{x})$ decreases fastest if one goes from $\mathbf{x}$ in the direction of the negative gradient of g at $\mathbf{x}$. It follows that, if $\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla g(\mathbf{x}_n)$ for a $\alpha \in \mathbb{R}^+$ small enough, then $g(\mathbf{x}_n) \geq g(\mathbf{x}_{n+1})$. In other words, the term $\alpha \nabla g(\mathbf{x})$ is subtracted from $\mathbf{x}$ because we want to move against the gradient, toward the local minimum. With this observation in mind, one starts with a guess $\mathbf{x}_0$ for a local minimum of g, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \cdots$ such that [1]

$$x_{n+1} = x_n - \alpha \nabla g(\mathbf{x}_n),\ n \geq 0. \tag{1}$$

The pseudo code for BGD is shown as follows:

Algo 1 Batch Gradient Descent

```
Input: $\alpha$: learning rate, $g(\theta)$: loss function
Output: $argmin_x g(x)$
1: function $GradientDescent(x_0, \alpha, step)$    # $x_0$: initial values
2:     for $i = 1, 2, \cdots, step$ do
3:         $gradient = \nabla g(x_0)$    # $\nabla$ denotes gradient operator
4:         $x = x - \alpha * gradient$
5:         $x_0 \leftarrow x$
6:     end for
7:     return $x$
8: end function
```

2.2 Accelerated Gradient Descent

Nesterov's Accelerated Gradient Descent (NGD or AGD for abbreviation) is a general approach that can be used to modify a gradient descent-type method to improve its initial convergence. Starting with the initial value x_0 and $y_0 = x_0$, AGD algorithm is iteratively defined by

$$x_k = y_{k-1} - \delta \nabla g(y_{k-1}),\ y_k = x_k + \frac{k-1}{k+2}(x_k - x_{k-1}), \tag{2}$$

where $g(\cdot)$ is the loss function, δ is a positive constant.

For these two equations, we could derive that:

$$\begin{aligned} \frac{x_{k+1}}{\sqrt{\delta}} &= \frac{y_k}{\sqrt{\delta}} - \sqrt{\delta}\nabla g(y_k) \\ \frac{x_{k+1}}{\sqrt{\delta}} &= \frac{x_k}{\sqrt{\delta}} + \frac{k-1}{k+2}\frac{x_k - x_{k-1}}{\sqrt{\delta}} - \sqrt{\delta}\nabla g(y_k) \\ \frac{x_{k+1} - x_k}{\sqrt{\delta}} &= \frac{k-1}{k+2}\frac{x_k - x_{k-1}}{\sqrt{\delta}} - \sqrt{\delta}\nabla g(y_k) \end{aligned} \tag{3}$$

The pseudo code for AGD is shown as follows:

2.3 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or

[1] https://en.wikipedia.org/wiki/Gradient_descent

Algo 2 Nesterov's Accelerated Gradient Descent

```
Input: $\alpha$: learning rate, $g(\theta)$: loss function
Output: $argmin_y g(y)$
    function $AcceleratedGD(x_0, y_0, \alpha, step)$    # $x_0, y_0$: initial values
        for $i = 1, 2, \cdots, step$ do
            $gradient = \nabla g(y_0)$    # $\nabla$ denotes gradient operator
            $x_1 = y_0 - \alpha * gradient$
            $y_1 = x_1 + \frac{i-1}{i+2}(x_1 - x_0)$
            $x_0 \leftarrow x_1,\ y_0 \leftarrow y_1$
        end for
        return $x_1, y_1$
    end function
```

subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof. [2]

The pseudo code for SGD is shown as follows:

Algo 3 Stochastic Gradient Descent

```
Input: $\alpha$: learning rate, $g(\theta)$: loss function
Output: $argmin_x g(x)$
    function $StochasticGD(x_0, \alpha, step)$    # $x_0$: initial values
        np.random.shuffle(training data)
        for example in (training data) do    # to obtain an approx. min
            for $i = 1, 2, \cdots, step$ do
                $gradient = \nabla g(x_0)$    # $\nabla$ denotes gradient operator
                $x = x - \alpha * gradient$
            end for
        end for
        return $x$
    end function
```

2.4 Accelerated Stochastic Gradient Descent

For this method, we apply Nesterov's acceleration scheme to stochastic gradient descent, which leads to the change of equation (2) as follows:

$$x_k^m = y_{k-1}^m - \delta \nabla \mathcal{L}^m(y_{k-1}^m; \mathbf{U}_{mk}^*),\ y_k^m = x_k^m + \frac{k-1}{k+2}(x_k^m - x_{k-1}^m), \tag{4}$$

with the initial values x_0^m and $y_0^m = x_0^m$, where $\mathcal{L}^n(\theta; \mathbf{U}_n) = \frac{1}{n}\sum_{i=1}^n \ell(\theta; U_i)$, $\ell(\cdot)$ is the loss function, $\mathbf{U}_n = (U_1, \cdots, U_n)'$ is a sample, U_is are i.i.d under

[2]https://en.wikipedia.org/wiki/Stochastic_gradient_descent

distribution Q, and $\mathbf{U}^*_{mk} = (U^*_{1k}, \cdots, U^*_{mk}),\ k = 1, 2, \cdots$ are independent mini-batches.

The pseudo code for ASGD is shown as follows:

Algo 4 Stochastic Gradient Descent

```
Input: $\alpha$: learning rate, $g(\theta)$: loss function
Output: $argmin_x g(x)$
  function $AcceleratedSGD(x_0, y_0, \alpha, step)$    # $x_0, y_0$: initial values
    np.random.shuffle(training data)
    for example in (training data) do    # to obtain an approx. min
      for $i = 1, 2, \cdots, step$ do
        $gradient = \nabla g(y_0)$    # $\nabla$ denotes gradient operator
        $x_1 = y_0 - \alpha * gradient$
        $y_1 = x_1 + \frac{i-1}{i+2}(x_1 - x_0)$
        $x_0 \leftarrow x_1,\ y_0 \leftarrow y_1$
      end for
    end for
    return $x_1, y_1$
  end function
```

2.5 Stochastic Gradient Descent with Momentum

In this paper, we also introduce one more SGD approach with a new term called "Momentum", which is similar to the concept of momentum in physics simulates the inertia of an object when it is moving, i.e., it retains the direction of the previous update to a certain extent when it is updated.

The Stochastic gradient descent with momentum remembers the update Δw at each iteration, and determines the next update as a linear combination of the gradient and the previous update:

$$\begin{aligned} \Delta w &:= \eta \Delta w - \alpha \nabla g_i(w) \\ w &:= w + \Delta w \end{aligned} \tag{5}$$

that leads to:

$$w := w - \alpha \nabla g_i(w) + \eta \Delta w, \tag{6}$$

where the parameter w which minimizes $g(w)$ is to be estimated, α is a step size (or the learning rate) and η is an exponential decay factor between 0 and 1 that determines the relative contribution of the current gradient and earlier gradients to the weight change. [3]

The pseudo code for SGD with Momentum is shown as follows:

[3] `https://en.wikipedia.org/wiki/Stochastic_gradient_descent`

Algo 5 Accelerated Stochastic Gradient Descent

```
Input: $\alpha$: learning rate, $\eta$: the exponential decay factor, $g(\theta)$: loss function
Output: $argmin_x g(x)$
    function $MomentumSGD(x_0, \alpha, \eta, step)$    # $x_0$: initial values
        np.random.shuffle(training data)
        for example in (training data) do    # to obtain an approx. min
            for $i = 1, 2, \cdots, step$ do
                $gradient = \nabla g(x_0)$    # $\nabla$ denotes gradient operator
                $x = x - \alpha * gradient + \eta * \Delta w$
            end for
        end for
        return $x$
    end function
```

3 Derivation for the Asymptotic ODE

3.1 ODE for Gradient Descent Algorithms

Since in the previous content, we already known that with initial value x_0 the plain gradient descent algorithm is iteratively defined by

$$x_k = x_{k-1} - \delta \nabla g(x_{k-1}),$$

Now we model $x_k, k = 0, 1, \cdots$ by a smooth curve $X(t)$ with the Ansatz $x_k \approx X(k\delta)$ and define a step function $x_\delta(t) = x_k$ for $k\delta \le t < (k+1)\delta$, then we could deduce that,

$$X(k\delta) = X((k-1)\delta) - \delta \nabla g\Big(X\big((k-1)\delta\big)\Big),$$

divided by δ, we get

$$\frac{X(k\delta) - X((k-1)\delta)}{\delta} + \nabla g\Big(X\big((k-1)\delta\big)\Big) = 0,$$

as $\delta \to 0$, $x_\delta(t)$ approaches $X(t)$ satisfying

$$\dot{X(t)} + \nabla g(X(t)) = 0,$$

where $\dot{X(t)}$ denotes the derivative of $X(t)$ and initial value $X(0) = x_0$.

3.2 ODE for Nesterov's Accelerated Gradient Descent

As mentioned before, with initial values x_0 and $y_0 = x_0$, Nesterov's accelerated gradient descent algorithm is iteratively defined by equation (2). To derive this ODE, we also do the similar operation as before. Define a step function $x_\delta(t) = x_k$ for $k\sqrt{\delta} \le t < (k+1)\delta$ and introduce the Ansatz $x_\delta(k\sqrt{\delta}) = x_k \approx$

$X(k\sqrt{\delta})$ for some smooth function $X(t)$ defined for $t \geq 0$. Then for the detailed process, we define:

$$z_k := \frac{x_{k+1} - x_k}{\sqrt{\delta}} \tag{7}$$

taking this equation into equation (3), we get

$$z_k = \frac{k-1}{k+2} \cdot z_{k-1} - \sqrt{\delta}\nabla g(x_k + \frac{k-1}{k+2}(x_k - x_{k-1})), \tag{8}$$

then substitute k with $k+1$, so we have

$$\begin{aligned} z_{k+1} &= \frac{k}{k+3} \cdot z_k - \sqrt{\delta}\nabla g(x_{k+1} + \frac{k}{k+3}(x_{k+1} - x_k)) \\ z_{k+1} &= (1 - \frac{3}{k+3})z_k - \sqrt{\delta}\nabla g(x_k + \frac{2k+3}{k+3}(x_{k+1} - x_k)) \\ \frac{z_{k+1} - z_k}{\sqrt{\delta}} &= -\frac{3}{k+3}\frac{z_k}{\sqrt{\delta}} - \nabla g(x_k + \frac{2k+3}{k+3} \cdot \sqrt{\delta} \cdot z_k). \end{aligned} \tag{9}$$

Taking $t = k\sqrt{\delta}$ and letting $\delta \to 0$ in equation (9), we obtain

$$\ddot{X}(t) + \frac{3}{t}\dot{X}(t) + \nabla g(X(t)) = 0, \tag{10}$$

with the initial conditions $X(0) = x_0$ and $\dot{X}(0) = 0$.

4 Asymptotic Convergence Result

In terms of Stochastic, we add n to the right corner of every notation to denote as the stochastic circumstance. As $\delta \to 0$ and $n \to \infty$, we have

$$sup_{t\in[0,T]}|x^n_\delta(t) - X^n(t)| = O_P(\delta^{1/2}|\log\delta|), \tag{11}$$

where $x^n_\delta(t)$ are the continuous-time step processes for discrete x^n_k, with continuous curves $X^n(t)$.

5 Implement Concrete Problems

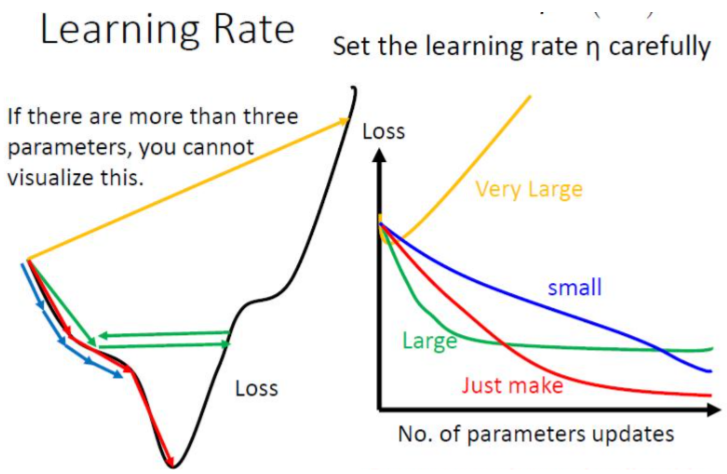
5.1 Some Toy Problems

Fundamentally, we will implement GD and SGD in some real problems starting from some toy ones:

Finding the minimum of one-dimensional function x^2 and the corresponding value of variable x.

As mentioned before, we apply GD and NGD to this question. To compare these two methods, we introduce one more parameters in these function: the threshold ϵ. Combined with the time function, we could calculate the procedure time under the same threshold based on different methods. Thus, we now have to pay more attention to these two variables: learning rate α and threshold ϵ.

Since we know that x^2 takes the minimum when $x = 0$, we let the initial value $x_0 = 10$ in GD and $x_0 = y_0 = 10$ in NGD to avoid procedure time being too short to compare to each other. Next, setting the value of α should be noticed. If α is too large, then we may not find the minimum, whereas the small α will also lead to the whole procedure too slow and there is no need for threshold. So, here we choose learning rate $\alpha = 0.001$.



Learning Rate
Set the learning rate η carefully
If there are more than three parameters, you cannot visualize this.
Loss
Loss
Very Large
small
Large
Just make
No. of parameters updates

Figure 1: Different Values of Learning Rate

For the next step, we have tried different values of threshold, ranging form $1.0E-3$ to $1.0E-20$, to see how threshold interfere with the frequency and time cost by different methods. The detailed information is shown as follows:

Threshold	Frequency		Time		Minimizer	
	GD	NGD	GD	NGD	GD	NGD
1.0E-03	4600	227	0.002	0.000	9.991E-04	-1.565E-04
1.0E-05	6900	719	0.002	0.001	9.9969E-06	-8.2152E-06
1.0E-10	12651	5635	0.005	0.003	9.9910E-11	-2.4731E-11
1.0E-15	18402	17503	0.01	0.015	9.9851E-16	2.0671E-17
1.0E-20	24152	29371	0.014	0.0239	9.9992E-21	4.6119E-21

Table 1: Implementing GD and NGD on 1-dimensional function based on different threshold

From the **Table 1**, we could observe that with threshold decreasing, the frequency and time increase obviously. While in comparison to each other, the situation changed between $\epsilon = 1.0E-10$ and $\epsilon = 1.0E-15$. NGD takes longer time than GD, which seems to be strange since NGD is the method that optimize the previous one. After inquiring professor Song, we state that too small threshold makes less sense and leads to some redundant and repetitive calculations, so it performs worse than GD.

Then we extend the objective function to two dimensions. Still from the most simple one:

$$f(\boldsymbol{\theta}) = \theta_1^2 + \theta_2^2,$$

we know that $f(\boldsymbol{\theta})$ takes the minimum when $\theta_1 = \theta_2 = 0$ since $f(\boldsymbol{\theta}) \geq 0$. We could see the figure of $f(\boldsymbol{\theta})$ in **Figure 2**.

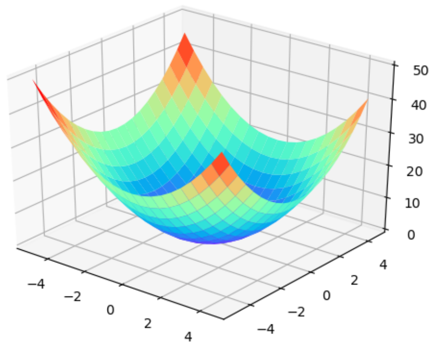


Figure 2: 2-dimensional Function $\theta_1^2 + \theta_2^2$

In order to be consistent and easy to observe, we still use the format of the previous table. Plus, we set the initial point $\boldsymbol{\theta} = [10, 10]$ and learning rate $\alpha = 0.001$.

Threshold	Frequency		Time		Minimizer	
	GD	NGD	GD	NGD	GD	NGD
1.0E-03	4773	438	0.0499	0.006	7.0665E-04	1.1677E-04
1.0E-05	7073	1913	0.0678	0.023	7.0705E-06	-3.4102E-06
1.0E-10	12824	8444	0.1386	0.1047	7.0663E-11	-3.3723E-11
1.0E-15	18575	17503	0.1855	0.2294	7.0621E-16	2.0671E-16
1.0E-20	24326	30003	0.2783	0.361	7.0579E-21	5.9257E-21

Table 2: Implementing GD and NGD on 2-dimensional function based on different threshold

From the **Table 2**, we could obviously become conscious of the similar situation as the previous one.

As for the next step, we will focus on SGD. Thus, we change the function one more time:

$$f(\theta) = \frac{1}{2}(f_1(\theta) + f_2(\theta)),$$

where $f_1(\theta) = \theta^2,\ f_2(\theta) = (\theta - 2)^2$.

Firstly, we derive the minimizer of this function mathematically:

$$\begin{aligned} f(\theta) &= \frac{1}{2}(f_1(\theta) + f_2(\theta)) \\ &= \frac{1}{2}(\theta^2 + (\theta - 2)^2) \\ &= \frac{1}{2}(2\theta^2 - 4\theta + 4) \\ &= \theta^2 - 2\theta + 2 \\ &= (\theta - 1)^2 + 1 \geq 1 \end{aligned} \tag{12}$$

Here, $f(\theta)$ takes the equal sign if and only if $\theta = 1$.

So we start to set the initial point $x_0 = 10$ in SGD and $x_0 = y_0 = 10$ in N-SGD with the same learning rate $\alpha = 0.001$. Because this method adopts the idea of randomness, the result is quite different from the previous one. It is not appropriate to set too big threshold like 0.1, so we start from $\epsilon = 0.01$ to $1.0E - 04$.

Threshold	Frequency		Time		Minimizer	
	SGD	N-SGD	SGD	N-SGD	SGD	N-SGD
0.01	6197	119	0.0369	0.0000	1.0098	0.9986
0.001	6423	637	0.0349	0.0049	1.0005	0.9992
1.0E-04	6696	1182	0.0389	0.0070	1.0000	0.9999

Table 3: Implementing SGD and N-SGD on 1-dimensional function based on different threshold

Up till now, the N-SGD seems to work pretty well, however, when $\epsilon = 1.0E - 04$, the frequency N-SGD costs varies from several hundreds times to the max step set initially. Generally speaking, we still could find the minimizer by using this method and take good advantage of randomness, so the time cost is far less than SGD. It is worthy of noting that SGD always takes more than 5000 times to get the minimizer no matter threshold is, while SGD can reduce the number of frequency times to less than 1000, which is a quite gap between these two methods. On the other hand, due to the randomness, we could not guarantee that N-SGD perform very well each time, just as mentioned before, not to say the smaller threshold. When $\epsilon = 1.0E-05$, the minimizer often could not be found based on N-SGD, or even though being found, the performance is poorer since frequency and time cost based on this method is larger than using SGD. Thinking over this case, we infer that the smaller threshold will restrict the performance of N-SGD. It has to find the point which meet the threshold requirement, but the calculation precision contradicts to this demand, so the procedure keeps running until the max step times.

What's more, for this problem, the learning rate α seems to play an important role especially for N-SGD. When adjusting α from 0.001 to 0.01, the frequency has decreased for SGD to several hundreds times, which is a great progress in contrast with the previous one shown in **Table 3**. On the other hand, it becomes more difficult to find the minimizer based on N-SGD.

To have a further study of this situation, we illustrate the phenomenon with one more example:

$$f(\theta) = \frac{1}{2}(f_1(\theta) + f_2(\theta)),$$

where $f_1(\theta) = -\theta^2,\ f_2(\theta) = 2(\theta - 1)^2$.

Similarly, we start the whole process from deriving the minimizer of this function mathematically:

$$\begin{aligned} f(\theta) &= \frac{1}{2}(f_1(\theta) + f_2(\theta)) \\ &= \frac{1}{2}(-\theta^2 + 2(\theta - 1)^2) \\ &= \frac{1}{2}(\theta^2 - 4\theta + 2) \\ &= \frac{1}{2}\theta^2 - 2\theta + 1 \\ &= \frac{1}{2}(\theta - 2)^2 - 1 \geq -1 \end{aligned} \tag{13}$$

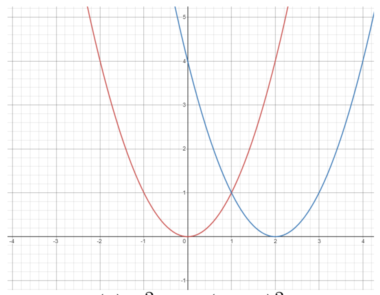
Here, $f(\theta)$ takes the equal sign if and only if $\theta = 2$.

Setting the initial point $x_0 = 10$ in SGD and $x_0 = y_0 = 10$ in N-SGD with the same learning rate $\alpha = 0.001$ as before. Plus, starting from $\epsilon = 0.01$ to $1.0E - 04$.

Threshold	Frequency		Time		Minimizer	
	SGD	N-SGD	SGD	N-SGD	SGD	N-SGD
0.01	8196	157	0.0479	0.0020	2.0097	1.9920
0.001	9999	505	0.0559	0.0020	2.0254	2.0009
1.0E-04	9999	9999	0.0539	0.0638	2.0819	-2.3430

Table 4: Implementing SGD and N-SGD on 1-dimensional function based on different threshold

In the frequency column of **Table 4**, 9999 is the max step, which represents that the procedure have not found the required minimizer until the loop ends. Here we could see that even SGD did not perform well under this circumstance. We deduce that this phenomenon is owing to the different direction of two parabolas, as shown in **Figure 3**, which leads to greater influence in the random process.


(a) θ^2 and $(\theta - 2)^2$

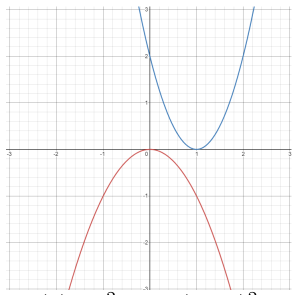

(b) $-\theta^2$ and $2(\theta - 1)^2$

Figure 3: Two Different Parabola Combinations

5.2 Rosenbrock Function

In mathematical optimization, the Rosenbrock function is a non-convex function, introduced by Howard H. Rosenbrock in 1960, which is used as a performance test problem for optimization algorithms.

The global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. To converge to the global minimum, however, is difficult.

The function is defined by

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

It has a global minimum at $(x, y) = (a, a^2)$, where $f(x, y) = 0$. [4]

In this section, we will take the specific circumstance that $a = 1$ and $b = 100$, i.e., the function is:

$$f(\boldsymbol{\theta}) = (1 - \theta_1)^2 + 100(\theta_2 - \theta_1^2)^2,$$

with the global minimum at $(1, 1)$. The plot is shown as follows,

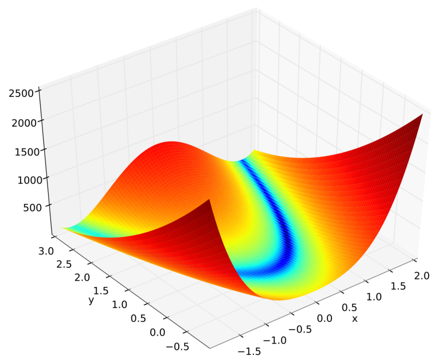


Figure 4: Rosenbrock Function when $a = 1,\ b = 100$

[4]`https://en.wikipedia.org/wiki/Rosenbrock_function`

At first, I have tried the same initial point $\boldsymbol{\theta} = [10, 10]$ as before, however, unless some specific learning rate and threshold, the overall performance did not seem to be well enough. Thus, for this problem, we set the initial point $\boldsymbol{\theta}$ to be $[0, 0]$ which is close to the global minimum $[1, 1]$.

For the next step, we have changed the value of learning rate α. Here is a new table reflects the trend of the "minimizer" we defined:

α	Minimizer				Frequency		Time	
	GD		NGD		GD	NGD	GD	NGD
	θ_1	θ_2	θ_1	θ_2				
0.0001	0.6737	0.4523	1.0000	1.0000	9999	1003	0.1964	0.0199
0.0002	0.9999	0.8221	1.0000	1.0000	9999	1414	0.1895	0.0309
0.0003	0.8925	0.7961	1.0000	1.0000	9999	3452	0.1895	0.0718
0.0004	0.9322	0.8686	1.0000	1.0000	9999	1248	0.2234	0.0240
0.0005	0.9562	0.9141	1.0000	1.0000	9999	2674	0.2025	0.0558
0.0006	0.9713	0.9433	1.0000	1.0000	9999	2238	0.2034	0.0519
0.0007	0.9810	0.9624	1.0000	1.0000	9999	1320	0.1835	0.0299
0.0008	0.9874	0.9749	1.0000	1.0000	9999	1411	0.2015	0.0320
0.0009	0.9916	0.9833	1.0000	1.0000	9999	2159	0.1865	0.0469
0.001	0.9944	0.9888	1.0000	1.0000	9999	1891	0.1985	0.0369
0.002	0.9999	0.9998	NAN	NAN	9999	9999	0.1935	0.2185
0.003	0.7393	0.5974	NAN	NAN	9999	9999	0.1845	0.2094
0.004	0.6420	0.3493	NAN	NAN	9999	9999	0.1975	0.1985
0.005	0.5342	0.2147	NAN	NAN	9999	9999	0.1885	0.1885
0.006	0.4452	0.1205	NAN	NAN	9999	9999	0.1984	0.2134
0.007	0.3655	0.0480	NAN	NAN	9999	9999	0.1865	0.2144
0.008	NAN	NAN	NAN	NAN	9999	9999	0.2044	0.1945
0.009	NAN	NAN	NAN	NAN	9999	9999	0.1875	0.1845
0.01	NAN	NAN	NAN	NAN	9999	9999	0.1985	0.1975

Table 5: Rosenbrock Function Minimizer Based on Two Methods

From **Table 5**, we could see the huge difference between these two approaches. The performance of GD is so poor that all frequency is up to 9999, the value of max step, no matter what α is, which indicates that the convergence efficiency is not good enough to reach the global minimizer $[1, 1]$. Besides, when $\alpha = 0.003$, the minimizer found by GD deviates from the accurate one $[1, 1]$. Even though the values of point have not shown **NAN**, these data make less sense. In addition, when α is big enough, we could not find the minimizer since there is a surge of value of two points. This phenomenon also happens when the initial points are far away from $[1, 1]$. In contrast, the performance of NGD is very impressive when $\alpha < 0.001$. It takes around 1500 times to reach to the minimizer. Plus, once the values of point are reasonable, the time cost by NGD is much shorter than by GD.

To understand this circumstance, we try to analyze this from the derivatives of Rosenbrock Function.

$$\begin{aligned}\frac{\partial f}{\partial \theta_1} &= 2(1-\theta_1)\cdot(-1) + 200(\theta_2-\theta_1^2)\cdot(-2\theta_1)\\ &= 2\theta_1 - 2 + 400\cdot\theta_1(\theta_1^2-\theta_2)\\ \frac{\partial f}{\partial \theta_2} &= 200\cdot(\theta_2-\theta_1^2)\end{aligned} \tag{14}$$

We could find that the coefficient is so large that a minor variation in α could greatly affect the values of these two gradient. If α increases 0.001, then the values of these derivatives are going to increase to a larger extent. Thus, we have to pay more attention to the value of α and initial points actually. Given that the original θ_1 and θ_2 are large, then the derivative will become too large within several calculation steps.

5.3 Calibrate the Black-Scholes Volatility

In this part, we are going to implement the Gradient Descent into Finance filed. One of the most famous and fundamental is the Black-Scholes Model.

5.3.1 Black-Scholes Model

The Black–Scholes (**BS** as abbreviation) or Black–Scholes–Merton model is a mathematical model for the dynamics of a financial market containing derivative investment instruments. From the partial differential equation in the model, known as the Black–Scholes equation, one can deduce the Black–Scholes formula, which gives a theoretical estimate of the price of European-style options and shows that the option has a unique price given the risk of the security and its expected return (instead replacing the security's expected return with the risk-neutral rate).[5]

The Black-Scholes formula calculates the price of European call and put options. The value of a call option for a non-dividend-paying underlying stock in terms of the BS parameters is given as follows:

$$\begin{aligned}C(S_t,t) &= N(d_1)S_t - N(d_2)Ke^{-r(T-t)}\\ d_1 &= \frac{1}{\sigma\sqrt{T-t}}\Big[\ln(\frac{\mathrm{S_t}}{\mathrm{K}}) + \big(\mathrm{r}+\frac{\sigma^2}{2}\big)(\mathrm{T}-\mathrm{t})\Big]\\ d_2 &= d_1 - \sigma\sqrt{T-t}\end{aligned} \tag{15}$$

where t is time, S_t is the price of the underlying asset at t, K is the strike price of the option, r is the annualized risk-free interest rate. $N(\cdot)$ denotes the standard normal cumulative distribution function, where $N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-z^2/2}\mathrm{dz}$, $\Phi(\cdot)$ is the pdf of standard normal distribution. σ is the measure of volatility.

[5] `https://en.wikipedia.org/wiki/Black-Scholes_model`

Plus, the price of a corresponding put option is:

$$P(S_t, t) = -N(-d_1)S_t + N(-d_2)Ke^{-r(T-t)}.$$

5.3.2 Calibration for the volatility

Take BS model as an example, we aim to find the minimizer of the following MSE loss:

$$f(\sigma) = \frac{1}{n}\Sigma_{i=1}^{n}(C_i(\sigma) - M_i)^2,$$

where σ is the volatility, $\{C_i(\sigma) : \ i = 1, 2, \cdots, n\}$ is the BS call prices for a stock with (S, T, K_i), and $\{M_i : \ i = 1, 2, \cdots, n\}$ is the market call prices for the same stock.

To find the minimizer, we need to calculate the derivatives of this function $f(\sigma)$:

$$\begin{aligned}
\frac{\partial f}{\partial \sigma} &= \frac{2}{n}\Sigma_{i=1}^{n}\left\{\Big(C_i(\sigma) - M_i\Big) \cdot \frac{\partial C_i(\sigma)}{\partial \sigma}\right\} \\
&= \frac{2}{n}\Sigma_{i=1}^{n}\left\{\Big(C_i(\sigma) - M_i\Big) \cdot \Big(S_t\frac{\partial N(d_1)}{\partial \sigma} - Ke^{-r(T-t)}\frac{\partial N(d_2)}{\partial \sigma}\Big)\right\} \\
&= \frac{2}{n}\Sigma_{i=1}^{n}\left\{\Big(C_i(\sigma) - M_i\Big) \cdot \Big(S_t\frac{\partial N(d_1)}{\partial d_1}\frac{\partial d_1}{\partial \sigma} - Ke^{-r(T-t)}\frac{\partial N(d_2)}{\partial d_2}\frac{\partial d_2}{\partial \sigma}\Big)\right\} \\
&= \frac{2}{n}\Sigma_{i=1}^{n}\left\{\Big(C_i(\sigma) - M_i\Big) \cdot S_t\Phi(d_1)\sqrt{T-t}\right\}
\end{aligned} \tag{16}$$

Then combined with **equation** (15), we could rewrite it as:

$$\frac{\partial f}{\partial \sigma} = \frac{2}{n}S_t\sqrt{T-t}\Sigma_{i=1}^{n}\left\{\Big(N(d_1)S_t - N(d_2)K_ie^{-r(T-t)} - M_i\Big) \cdot \Phi(d_1)\right\} \tag{17}$$

Therefore, **equation** (17) is the gradient calculated what we need in GD and NGD. Take this into the previous **Algo 1** and **Algo 2**, we will be able to find the minimizer. That is to say, the objective that calibrating the volatility could be finished by making use of it.

6 Conclusion

In this article, we aim to study the Gradient Descent, Nesterov Gradient Descent and Stochastic Gradient Descent, which are often used to find the minimizer. Thus we start the research from some toy problems, like simple 1-dimensional and 2-dimensional functions. By comparing the performances of these methods in the concrete problems, we find that all parameters, α, ϵ, x_0

and steps, are crucial to find the minimizer. We need to pay much attention to select the appropriate ones, especially in a more complex situation.

To sum up, Gradient Descent uses all the training data for each update to minimize the loss function. If there is only one minimum value, then GD considers all the data in the training set and iteratively moves towards the minimum value, but the disadvantage is that if the sample value is large, the update speed will be very slow, which has been shown in the 2-d function example.

Stochastic gradient descent only considers one sample point in each update, which will greatly speed up the training data. This has just overcome the disadvantage of GD. However, due to the large amount of noise points in the training data, it is not necessarily to update towards the minimum value in each update process using noise points, as the 1-d function $f = f_1 + f_2$ shown before. Thanks to the multiple rounds of update, the overall direction of SGD algorithm is roughly updated towards the minimum value, which still improves the speed to some extent.

Nesterov Gradient Descent is actually a kind of momentum. When calculating the gradient, Nesterov first updates the parameters with the current velocity v, and calculates the gradient with the updated temporary parameters. When combined with GD, Nesterov improves the error convergence from $O(1/k)$ to $O(1/k^2)$, while it has no improvement associated with SGD.

7 Reference

(1) Yazhen Wang, Shang Wu, Asymptotic Analysis via Stochastic Differential Equations of Gradient Descent Algorithms in Statistical and Computational Paradigms, Journal of Machine learning research 2020.

(2) Weijie Su, Stephen Boyd, Emmanuel Candes, A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights, Journal of Machine learning research 2016.

(3) Zhuang Yang, Cheng Wang, Zhemin Zhang, Jonathan Li, Accelerated Stochastic Gradient Descent with Step Size Selection Rules, Signal Processing 159 (2019) 171-186.

(4) https://github.com/Jun-629/Capstone-Stochastic-Gradient-Descent/tree/main