Explained and Built Gradient Descent Algorithm in R (English Version)

目錄

1. Basic Idea：

To find the minimum value of an arbitrary function, we have to find the point with zero derivative. However, most programming languages cannot solve the equation directly, so we use an algorithm called "gradient descent" to find this point with the gradient of that arbitrary function.

2. Formula：

In this algorithm, we will use gradient (i.e. the partial derivative with respect to Xi) to find the minimum value of a function. Here we take a 2-dimensional function as an example:

As the graph shows below, when x is at X_1, the minimum of the function, X_n, is at the right-hand side of X_1. Therefore, the next iteration, X_2, has to shift to the right($X_1 < X_2$). Since the gradient of the function at X_1 is smaller than 0, it is natural to choose X_2 to satisfy $X_2 = X_1 - \frac{\partial f}{\partial x}$. However, $\frac{\partial f}{\partial x}$ may be too large or too small relative to x, so it may take too much time for the algorithm to find the minimum value of the function. Therefore, to avoid this issue, we have to control the extent to which X moves each iteration, which gives us the following equation:

$$X_{j+1} = X_j - \eta\frac{\partial f}{\partial x}$$

If we expand this equation to n-dimension, we will have:

$$X_{1,j+1} = X_{1,j} - \eta\frac{\partial f}{\partial x_1}$$

$$X_{2,j+1} = X_{2,j} - \eta\frac{\partial f}{\partial x_2}$$

.

.

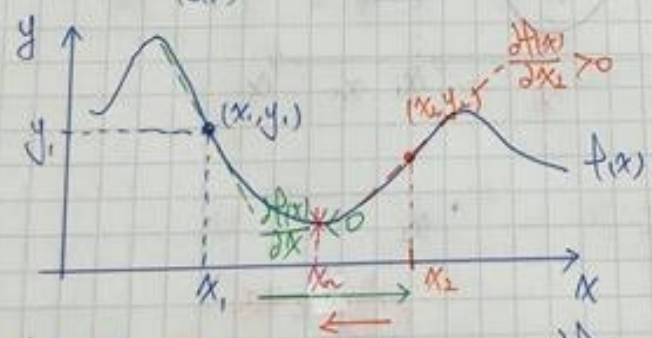
.

$$X_{n,j+1} = X_{n,j} - \eta\frac{\partial f}{\partial x_n}$$

Typically, η is between 0 and 1 and is called the "learning rate".

Gradient descent: (Assume finding Minimum)

(GD)



① 隨意選一個點, $x_1 \Rightarrow \frac{\partial f(x)}{\partial x_1} < 0 . \Rightarrow \nabla f(x) = -\frac{\partial f(x)}{\partial x_1} > 0$

x_1 往 + 方向走.

$x_2 \Rightarrow \frac{\partial f(x)}{\partial x_2} > 0 \Rightarrow \nabla f(x) = -\frac{\partial f(x)}{\partial x_2} < 0$

x_2 往 − 方向走

⋮

until converges. to x_n

take Linear Regression as example:

$Y = w_1 X$, $(Y=10,\ X=5,\ w_1 \text{ correct} = 2)$

$\mathcal{L} = MSE = (Y-\hat{Y})^2$

iter 1:

$w_1^{(0)} = 0 \Rightarrow \hat{Y} = 0 \Rightarrow \mathcal{L} = (10-0)^2 = 100$

$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_1} = 2(Y-\hat{Y}) \times (-1) \times X$

$= 2 \times 10 \times (-1) \times 5$

$= -100$

Let $\eta = 0.01$,

$w_1^{(1)} = w_1^{(0)} - \eta \times \frac{\partial \mathcal{L}}{\partial w_1}$

$= 0 - 0.01 \times (-100)$

$= 1$.

iter 2:

$w_1^{(1)} = 1 \Rightarrow \hat{Y} = 5 \Rightarrow \mathcal{L} = (10-5)^2 = 25$

$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_1} = -2 \times 5 \times 5 = -50$

$w_1^{(2)} = w_1^{(1)} - \eta \times \frac{\partial \mathcal{L}}{\partial w_1}$

$= 1 - 0.01 \times (-50) = 1.5$

iter 3:

...

until $\frac{\partial \mathcal{L}}{\partial w_1} = 0$.

3. Problems of the Gradient Descent：

1. If the function is not differentiable, then this algorithm is not implementable.

2. If the chosen initial value (X_1) is with 0 derivative, then $X_n = X_{n-1} = \cdots = X_1$.

3. If the function has multiple local minimums, then it is probable that we are not able to find the global minimum because of the inappropriate initial value. To solve this, we can randomly draw several initial values and compare the values they converge to find the global minimum.

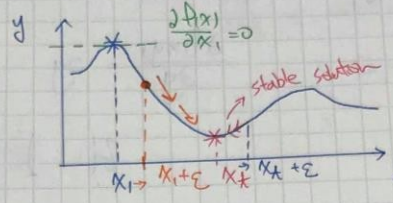
4. If $\frac{\partial f}{\partial x}$ is too large, then it is possible that X_n will diverge, or it will be very difficult for the algorithm to find the minimum value.

Facing Problem:

① Stochastic point = Max or Min (i.e. $\frac{\partial f(x)}{\partial x} = 0$)

Sol. ⇒ test if it's a stable dynamic system solution.

$x^{(t)} = x^{(t-1)} + \varepsilon$, $\varepsilon \ll$ (變化量)

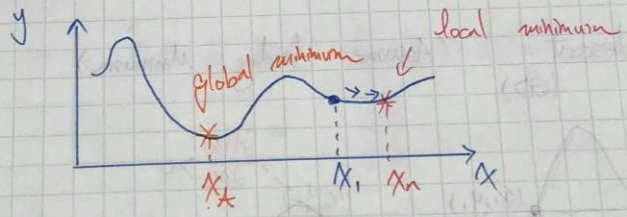


if choose Max ⇒ system solution will change

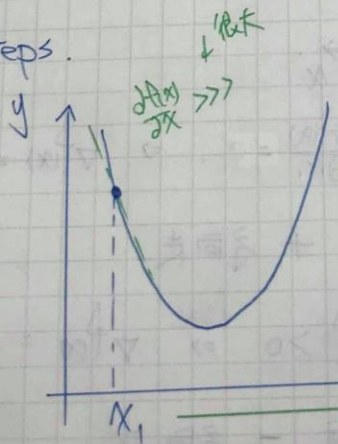
Min ⇒ system solution unchanged.

② local minimum problem.

(Hard to find global minimum)



③ Steps.



x_2

$\left(\frac{\partial f(x)}{\partial x_2} \times \text{變化}\right)$

..... diverge.

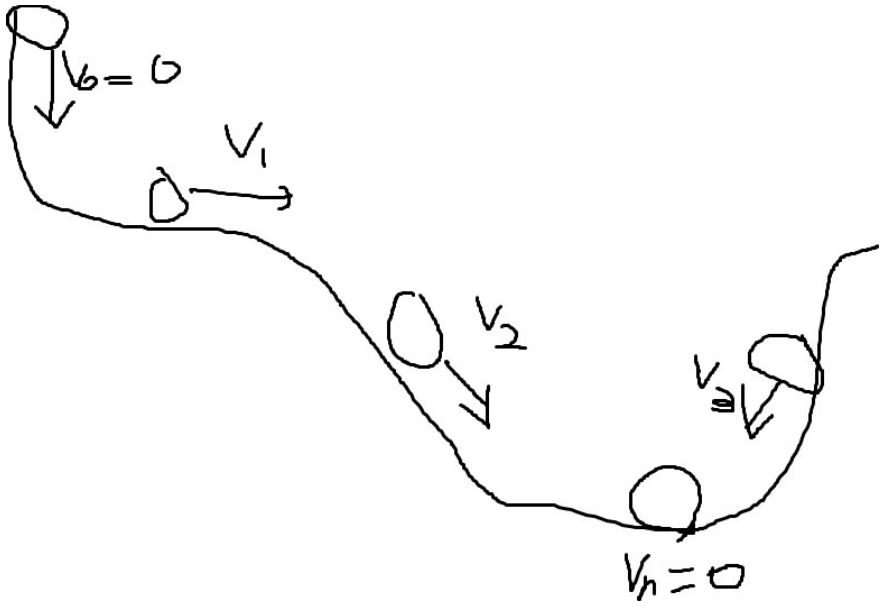
Sol. η (learning rate), $x^{(t)} = x^{(t-1)} - \eta \frac{\partial f}{\partial x}$

small, Ex: 0.000

4.Method to Enhance Gradient

1.Momentum Descent:

This algorithm aims to solve the issue of finding the local minimum rather than the global minimum. As shown by the graph below, imagine that there is a slide, and we put a ball at the top of that slide with zero velocity. When touching a flat ground (i.e. a local minimum), the ball will keep moving with velocity V_1 because of inertia. Therefore, the ball may not get stuck in the local minimum and therefore converge to the global minimum.



To simulate inertia in the computer, we create a new variable V, the value of which should increase at a descending slope while that of which should decrease at an increasing slope. To ensure that V will not inflate too fast, we add a constant in front of V_j, so we have:

$$V_{j+1} = \beta V_j - \eta \frac{\partial f}{\partial x_i}$$
$$X_{1,j+1} = X_{1,j} + V_{j+1}$$
$$X_{2,j+1} = X_{2,j} + V_{j+1}$$
$$\vdots$$
$$X_{n,j+1} = X_{n,j} + V_{j+1}$$

Note that β, η are in [0,1], and β usually is 0.9

2. Adagrad Gradient Descent：

In the original algorithm, it is possible that the velocity of X is too fast so that it cannot converge. Therefore, to slow it down, we create a new variable n such that

$$n = \sum_{r=1}^{j+1} \left(\frac{\partial f}{\partial x_{i,r}} \right)^2$$

and

$$X_{i,j+1} = X_{i,j} - \eta * \frac{1}{\sqrt{n+\epsilon}} \frac{\partial f}{\partial x_i}$$

$$where\ \epsilon\ is\ a\ very\ small\ positive\ number$$

3. Adam ：

This algorithm combines Momentum and Adagrad Gradient Descent, and the following are common formula for this algorithm:

$$V_{j+1} = \alpha V_j + (1-\alpha) \frac{\partial f}{\partial x_i}$$

$$N_{j+1} = \beta N_j + (1-\beta) \left(\frac{\partial f}{\partial x_i} \right)^2$$

$$X_{i,j+1} = X_{i,j} - \eta * \frac{V_{j+1}}{\sqrt{N_{j+1}+\epsilon}}$$

Note that the first equation is the idea of Momentum and the second is the idea of AGD.

Usually, we set $\alpha = 0.9$ and $\beta = 0.999$.

5. Implement the Basic Method of Gradient Descent:

```
#derivative

library(stats) #import package "stats"
f1.exp <- expression(x^2 + 3*x +1) # define the function we want to
differentiate dx1 <- D(f1.exp,"x") # first derivative
x <- 1:5 #define x
eval(dx1) #get value
dx2 <- D(dx1, "x") # second derivative
eval(dx2) #get value

#partial derivative

f2.exp <- expression(x^2*y+y^2) #define the function we want to
differentiate dx1 <- D(f2.exp,"x") #first partial derivative with respect to
x
dy1 <- D(f2.exp,"y") #first partial derivative with respect
to y x <- 1:5 #define x
y <- 2:6 #define y
eval(dx1) #get value
eval(dy1)
#if length(x)/length(y) ==
integer x <- 1:5
y <- 1:10
eval(dx1) #calculable
#if length(x)/length(y) != integer
eval(dy1)
x <- 1:5
y <- 2:5
eval(dx1)#calculable, but a warning message appears
```

```
eval(dy1)
#the shoter vector will repeat


#gradient descent


#2-dimentional case
#find the minimum value of f(x) = 5*x^6-3*x^3-2*x^2+x-10


f <- expression(5*x^6-3*x^3-2*x^2+x-10) #define function
# define gradientdescent function, where gamma is our learning rate(0 to 1 usually), and iter
is how many times we try to find our minimum.
gradientDesc_D2 <- function(f,initial_value,gamma,iter)
{
  l <- gamma
  dx <- D(f,"x") #find derivative
  result <- numeric(length = length(initial_value)) #create a vector to store results
  x_ans <- numeric(length = length(initial_value)) #create a vector to store x
  for (i in 1:length(initial_value)) {
    x <- initial_value[i] #take initial
    x for (j in 1:iter) {
      x <- x-l*eval(dx) #find min x
    }
    result[i] <-
    eval(f) x_ans[i]
    <- x
  }
  ans <- min(result) #find min value
  x <- unique(round(x_ans[grep(ans,result)],10)) #find x value when the function is
  minimized if (is.nan(ans) == TRUE){
    print("The minimum of the function does not exist")
  }else{
      paste("The minimum of the function is",ans,"at postion x =", x , sep = " ")
```

```
  }

}

#test
set.seed(100)
x <- runif(3,0,1)
gradientDesc_D2(f,x,0.12,500)
#check
fx <- function(x){
return(5*x^6-3*x^3-2*x^2+x-10)
}
curve(fx,-1,1)
#we find that Gradint Descent only found local
minimum #try to adjust our learning rate
gradientDesc_D2(f,x,0.0012,500)
#we find the global minimum
#we can alter any parameter to find global minimum

#3-dimentional case

#find the minimium of f(x,y) = (x+y)^2-(x+y)-2

f <- expression((x+y)^2-(x+y)-2)

gradientDesc_D3 <- function(f,initial_value_x,initial_value_y,gamma,iter)
{
  l <- gamma
  dx <- D(f,"x") #find partial derivative with respect to
  x dy <- D(f,"y") #find partial derivative with respect
  to y
```

```
  result <- numeric(length = length(initial_value_x)) #create a vector to store
  results x_ans <- numeric(length = length(initial_value_x)) #create a vector to
  store x
  y_ans <- numeric(length = length(initial_value_y))
  for (i in 1:length(initial_value_x)) {
    x   <-   initial_value_x[i]   #take
    initial  x  y  <-  initial_value_y[i]
    #take initial y for (j in 1:iter) {
      x_new  <-  x  -  l*eval(dx)  #store  new-found  x
      temporarily  y  <-  y  -  l*eval(dy) #find  min  x  and
      min y
      x <- x_new
    }
    result[i] <-
    eval(f) x_ans[i]
    <- x y_ans[i] <-
    y
  }
  ans <- min(result) #find min
  value x <-
  x_ans[grep(ans,result)]
  y <- y_ans[grep(ans,result)]#find x value when the function is
  minimized if (is.nan(ans) == TRUE){
    print("The minimum of the function does not exist")
  }else{
    paste("The minimum of the function is",ans,"at postion x =", x ,", y =",y, sep = " ")
  }

}
#test
set.seed(10)
x <- runif(10,-1,1)
```

```
y <- runif(10,-1,1)
gradientDesc_D3(f,x,y,0.2,50000) #minimum at several (x,y)
```

6.Question:

試著以上述Momentum的方式改變 gradientDesc_D3 function 裡的程式，然後找出例題中函數

（(x+y)^2-(x+y)-2）的最小值吧！

7.Reference:

[1] Sebastian Ruder. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747v2, 2017.
[2] Diederik P. Kingma, Jimmy Lei Ba. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. arXiv preprint arXiv:1412.6980v9, 2017.