Gradient descent

JrPhy

Introduction

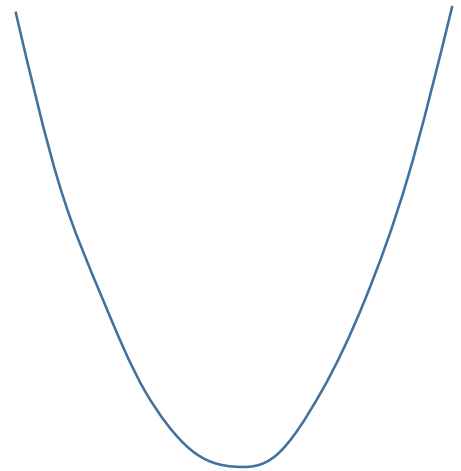
- Artificial Intelligent (AI) is famous today, and its fundamental algorithm is gradient descent, the other algorithms base on this method, like conjugate gradient descent, stochastic gradient descent, ada gradient descent, etc.
- The core idea of gradient descent is finding the minimum, it can be found by solving normal equation then uses QR factorized. Although normal equation gets global minimum definitely, it spends much time, so AI uses gradient descent to modify for the efficiency and accuracy.

Concept

- For a square equation, there is a global maximum or minimum. We can use derivative to get it.
- $f(x) = 2x^2+4x+3$ → $f\,'(x) = 4x+4 = 0$ minimum is (-1,1)
- Since computer can't compute derivative, so we use finite difference.

$$f'(x) = \frac{df(x)}{dx} \approx \frac{\Delta f(x)}{\Delta x} = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

- Here we want to solve the x^* such that $f(x^*)$ is minimum

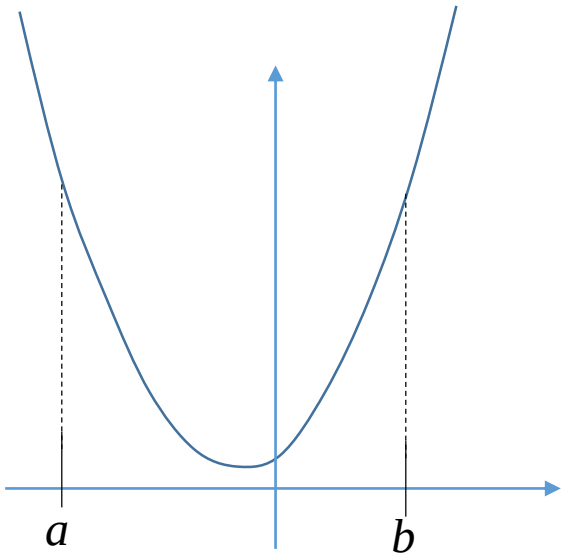


Concept

- 1st we guess a x as a start point, then use $f\,'(x)$ to find x^*
- If we start from a, and $a < 0$, then $f\,'(a) = 4a+4 < 0$, by the figure, we know the answer is in the right of a, so we minus it and

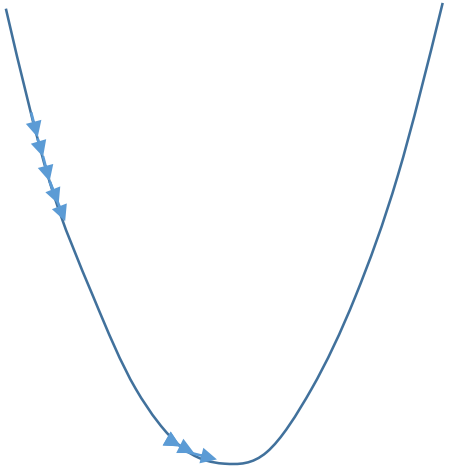
$$x_{n+1} = x_n - \lambda f\,'(x_n)$$

- That is the idea of gradient descent.
- In computer, Δx can not be as small as you want, it's a fix number, so we times a constant λ to control the step.
- We choose a initial point stochastically, so it calls Stochastic Gradient Descent (SGD)

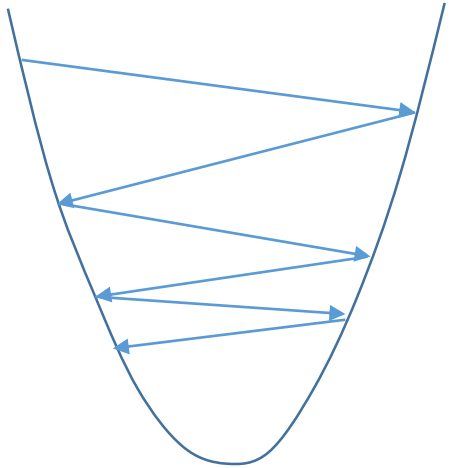


Concept

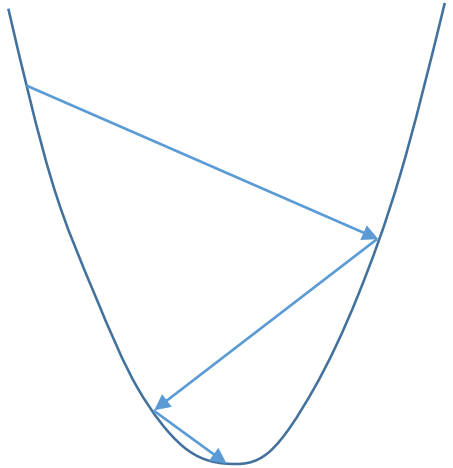
- Here times a constant λ to try the convergent rate, it's called learning rate, there is a best constant or function that convergent rate is fastest.
- too slow



may not converge



the best



Multi-dimension gradient descent

- Suppose the function f is $z = f(x,y)$, then the gradient is

$$\nabla f(x,y) = \frac{\partial}{\partial x} f(x,y)\hat{x} + \frac{\partial}{\partial y} f(x,y)\hat{y} = f_x(x,y)\hat{x} + f_y(x,y)\hat{y}$$

$$\begin{Bmatrix} x_{n+1} \\ y_{n+1} \end{Bmatrix} = \begin{Bmatrix} x_n \\ y_n \end{Bmatrix} - \eta \begin{Bmatrix} f_x \\ f_y \end{Bmatrix}$$

- Like do 2 times 1d gradient descent.
- Just mimic the 2d version to extend to multi-dimension.

Example

- $y = f(x) = x^2 - 4x + 2$, global minimum at $x = 2$
- $z = f(x, y) = x^2 - 4x + 2 + y^2 - 4y + 2$, global minimum at $x = 2$, $y = 2$.
- You can try my code to change x_0, y_0 and λ to see the iterators times. The error is smaller, the result is more precise, but it needs more iteration.

limitation

- For a quadric polynomial, there are 2 local minimum, then how can we know the result is global minimum?
- The answer is **NO**! because 1st derivative can just find the **local minimum**(maximum), its it limitation.
- And for multi variable, there exists **saddle point**, this algorithm may stop there, so there are other methods to avoid this issue.

Multi-dimension gradient descent

- Suppose the function f is $z = f(x,y)$, then the gradient is

$$\nabla f(x,y) = \frac{\partial}{\partial x} f(x,y)\hat{x} + \frac{\partial}{\partial y} f(x,y)\hat{y} = f_x(x,y)\hat{x} + f_y(x,y)\hat{y}$$

$$\begin{Bmatrix} x_{n+1} \\ y_{n+1} \end{Bmatrix} = \begin{Bmatrix} x_n \\ y_n \end{Bmatrix} - \eta \begin{Bmatrix} f_x \\ f_y \end{Bmatrix}$$

- It like does 2 times 1d gradient descent.

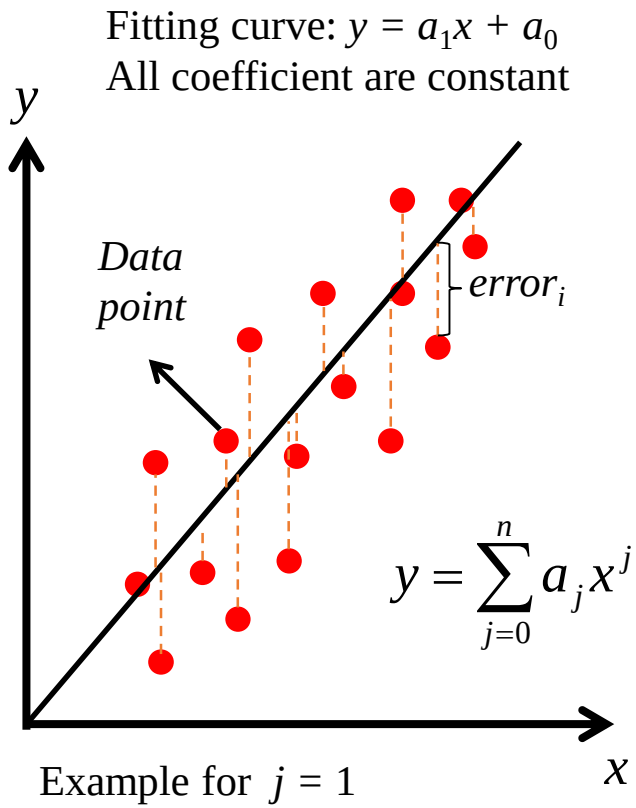
Linear regression by gradient descent

- **Concept**: There are many data (x_i, y_i), after setting on the x-y plane, the distribution may be fitted by some curve $y = f(x)$ or $f(x, y) = k$, k is some constant, $\forall\, x, y, k \in \mathbb{R}$.

- Define $error_i = y_i - y$

- Total error $E = \sum_{i=1}^{n} (y_i - (bx_i + a))^2$

- **Want**: E is minimum value.



Linear regression by gradient descent

- Find the derivative, here are 2 independent variable, so take the partial derivative(the result is the same as the total derivative)

$$\frac{\partial E}{\partial b} = \frac{1}{n}\sum_{i=1}^{n} -2(x_i)(y_i - bx_i - a) \quad \frac{\partial E}{\partial a} = \frac{1}{n}\sum_{i=1}^{n} -2(y_i - bx_i - a)$$

- Divide by n because of n data. Then apply the algorithm

$$b_{i+1} = b_i - \lambda \frac{1}{n}\sum_{i=1}^{n} -2(x_i)(y_i - bx_i - a) \quad a_{i+1} = a_i - \lambda \frac{1}{n}\sum_{i=1}^{n} -2(y_i - bx_i - a)$$

- λ is the step size determined by you.

Polynomial regression by gradient descent

- Most of the procedure are the same as linear regression, only the error is different.
- Suppose we want to regress a polynomial with degree *m*, then

$$y = \sum_{i=0}^{n} a_i x^i \qquad E = \sum_{j=1}^{n} (y_j - (\sum_{i=0}^{m} a_i x^i))^2 \qquad \frac{\partial E}{\partial a_i} = \frac{\lambda}{n} \sum_{j=1}^{n} 2(y_j - (\sum_{i=0}^{m} a_i x^i))$$

- Here I use superscript *k* to represent the next coefficient

$$a_2^{k+1} = a_2^k - \frac{\lambda}{n} \sum_{j=1}^{n} 2x^2 (y_j - (\sum_{i=0}^{2} a_i^k x^i))$$

Polynomial regression by gradient descent

- Here you should be careful about the numerical problem, we take power of 2 as example. Its formula is

$$a_2^{k+1} = a_2^k - \frac{\lambda}{n}\sum_{j=1}^{n} 2x^2(y_j - (\sum_{i=0}^{2} a_i^k x^i))$$

- and its degree is 4, that means if you use a polynomial with its degree bigger than 2, you have to use much smaller λ to ensure it will converge.

Adaptive gradient descent (Adagrad)

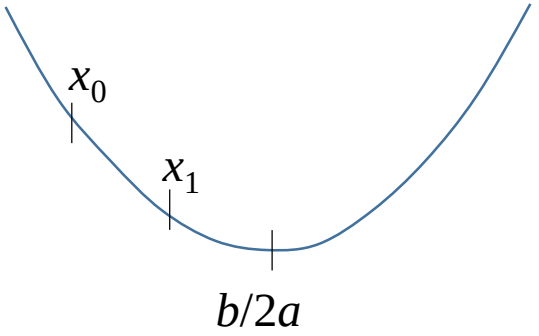
- In SGD, the learning rate is fixed, so the convergent rate depends on the learning rate. At first, the learning rate is big, then in the iteration, the learning rate is smaller than the beginning, because it's closer to the target.
- A mothed for adaptive the learning rate in each step, is Adagrad.

$$w^{k+1} = w^k - \frac{\eta}{\sqrt{\sum_{i=1}^{k} \left(g^i\right)^2}}$$

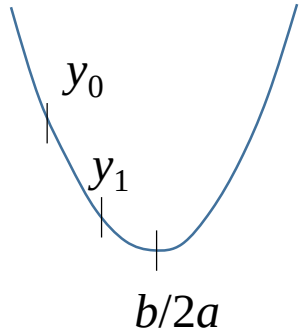
$$w^{k+1} = w^k - \frac{\eta^k}{\sigma^k} g^k \qquad \eta^k = \frac{\eta}{\sqrt{k+1}} \qquad g^k = \frac{\partial f(\theta^k)}{\partial w} \qquad \sigma^k = \frac{1}{\sqrt{k+1}} \sqrt{\sum_{i=1}^{k} \left(g^i\right)^2}$$

Adaptive gradient descent (Adagrad)

- For a parabolic function $f(x) = ax^2+bx+c$, $a > 0$, the $x^* = b/2a$. The initial guess is x_0, so the step bases on how far to the x^*, the farther distance, the bigger step, and the 1st derivative is bigger, too.



- Then how about multi-dimension function? Consider $f(x, y)$, obviously
- $\left.\frac{\partial f(x,y)}{\partial x}\right|_{x=x_0} > \left.\frac{\partial f(x,y)}{\partial x}\right|_{x=x_1}$ and $\left.\frac{\partial f(x,y)}{\partial y}\right|_{y=y_0} > \left.\frac{\partial f(x,y)}{\partial y}\right|_{y=y_1}$



- But $\left.\frac{\partial f(x,y)}{\partial y}\right|_{y=y_0} > \left.\frac{\partial f(x,y)}{\partial x}\right|_{x=x_0}$ $y_0 > x_0$

Adaptive gradient descent (Adagrad)

- So that we should divide by the 2nd derivative that it shows the correct distance to the minimum.

$$\left| x_0 + \frac{b}{2a} \right| = \left| \frac{2ax_0 + b}{2a} \right| = \left| \frac{\partial f(x) / \partial w|_{x=x_0}}{\partial^2 f(x) / \partial w^2 |_{x=x_0}} \right|$$

- But calculate 2nd derivative need much time cost, so we use the summation of the 1st derivative to approximate it.