Fakultas Informatika
School of Computing
Telkom University

Regression

- Gradient Descent

DAY

Review of Problem

Problem

- “JalanJalan” adalah **perusahaan travel** yang sangat sering menggunakan jasa go-car untuk mengantar tamu-tamu mereka. Karena mereka harus menghitung anggaran bulanan, maka mereka harus **menghitung perkiraan biaya Go-car** jauh hari sebelum digunakan, sehingga pengecekan langsung pada aplikasi go-car dianggap bukan cara yang tepat.
- “JalanJalan” memiliki DATA history biaya go-car selama beberapa bulan terakhir.
- Pertanyaan: Bagaimana cara kita untuk menentukan MODEL dari data history biaya go-car untuk memprediksi tarif bagi perusahaan “JalanJalan” tersebut?

*Adopted from Machine Learning course IF-TelU

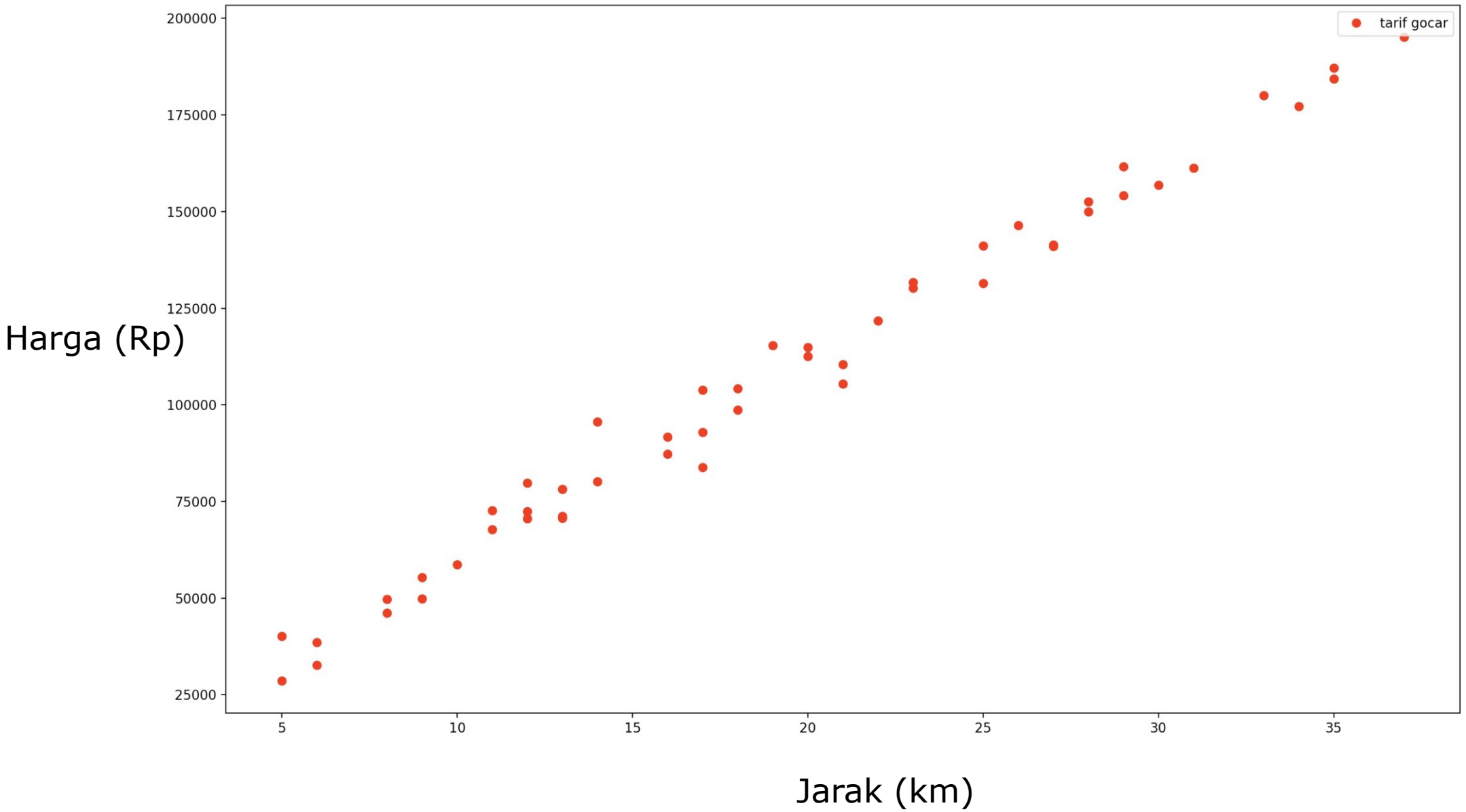
DATA historis Go-car

- Sebagai contoh data go-car dari 15 kali naik Go-car sebagai berikut:

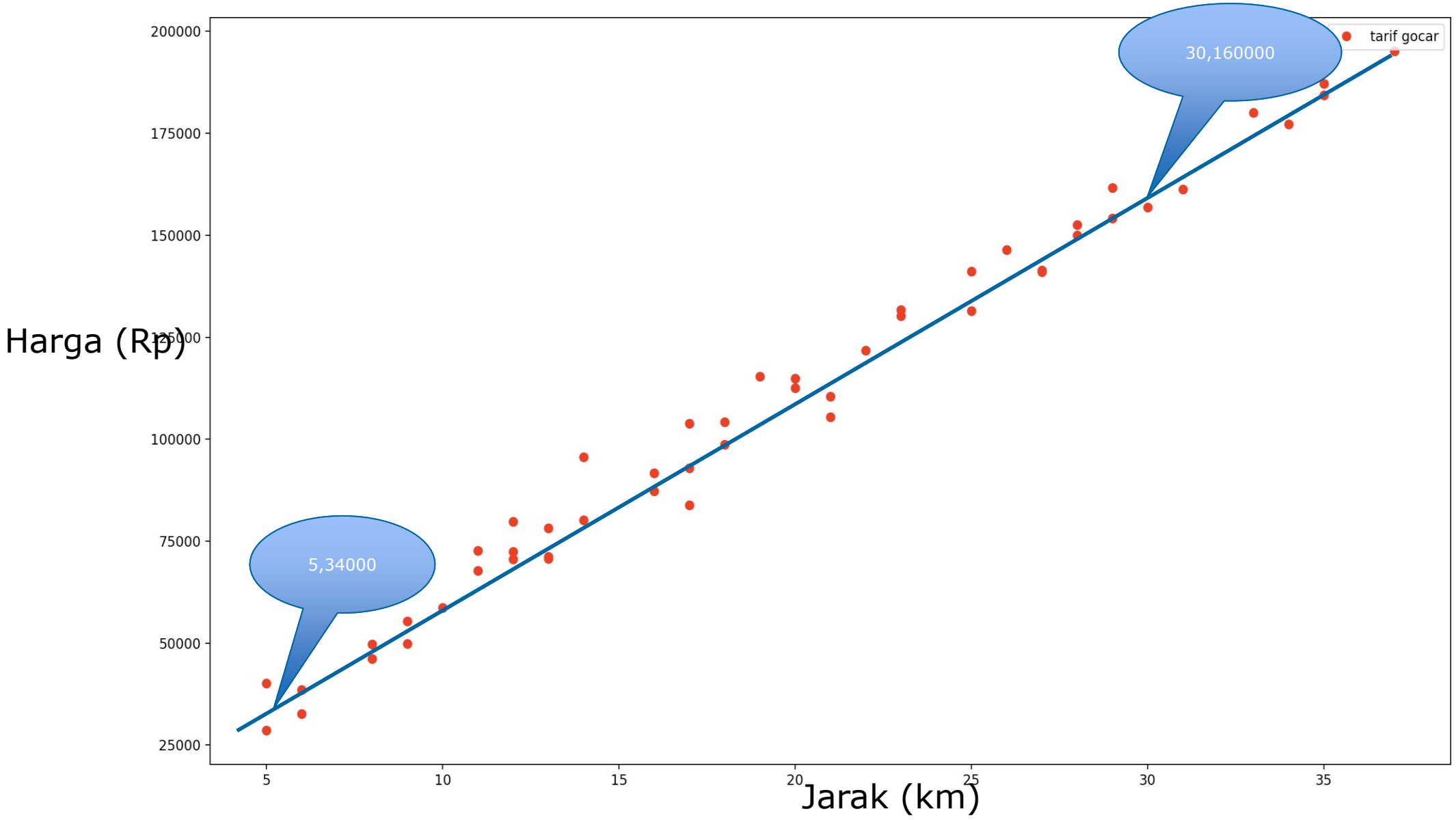
- What can you do with this history data ?

NO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
KM	5	5	6	6	8	8	9	9	10	11	11	12	12	12	13
TARIF	28600	40100	32600	38600	49700	46100	55300	49800	58700	67700	72600	79800	72400	70600	70700

Data plot


tarif gocar
200000
175000
150000
125000
100000
75000
50000
25000
5
10
15
20
25
30
35
Harga (Rp)
Jarak (km)

Model → GARIS !


tarif gocar
30,160000
5,34000
Harga (Rp)
Jarak (km)
200000
175000
150000
125000
100000
75000
50000
25000
5
10
15
20
25
30
35

Persamaan Garis?

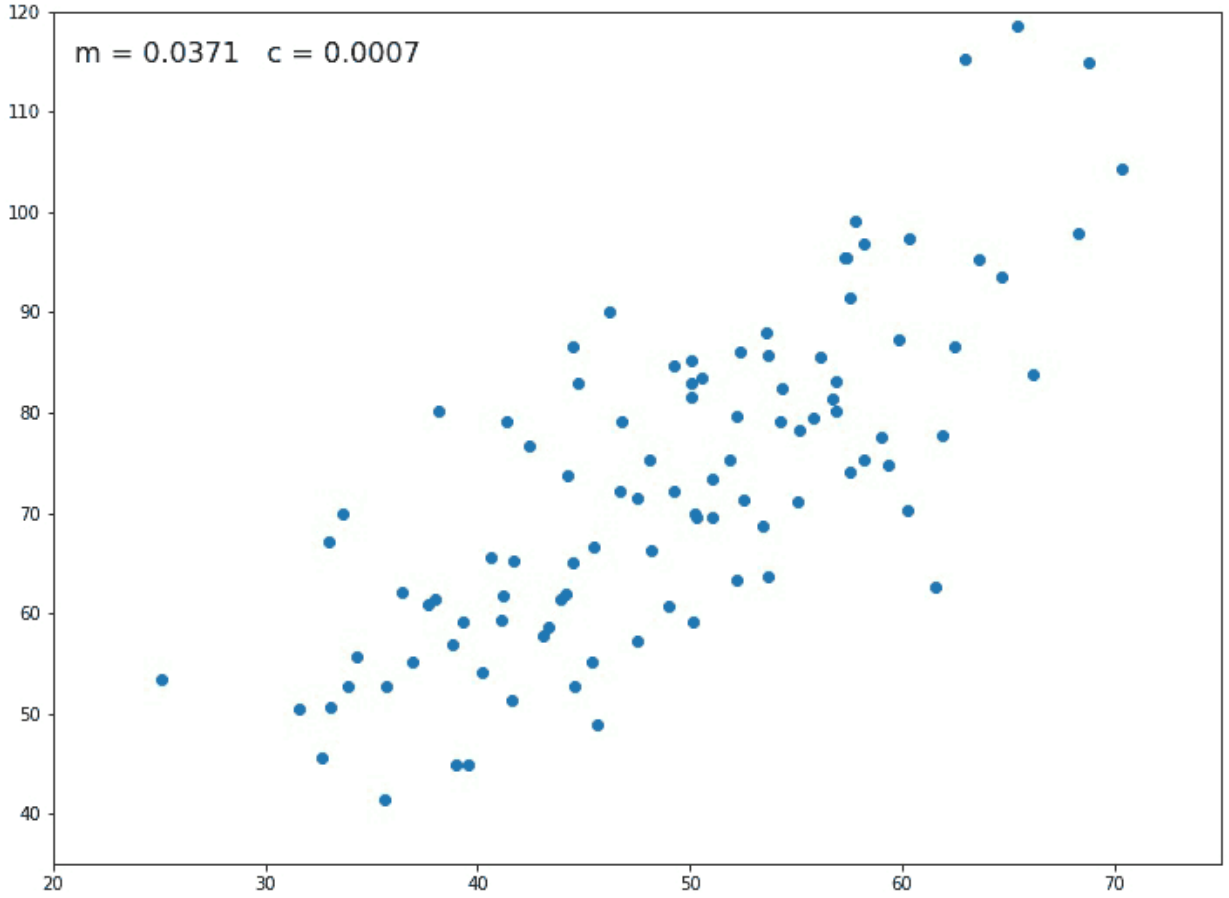
- **y = a + mx**
- **Cari m**
- m = slope = (y2-y1)/(x2-x1)
- = (160000-34000)/(30-5)
- = 126000/25 = 5040

- **Cari Persamaan Garis dari 1 titik**
- m = (y-y1)/(x-x1)
- y = y1+m(x-x1)
- y = 34000+5040(x-5)
- y = 34000 + 5040x – 25200
- **y = 8800 + 5040x**

Dua komponen utama pembentuk Model (yaitu GARIS) adalah:

- m (gradient)
- a (konstanta)

Linear Regression: Gradient Descent



How can we find “best line” by using Gradient Decent?

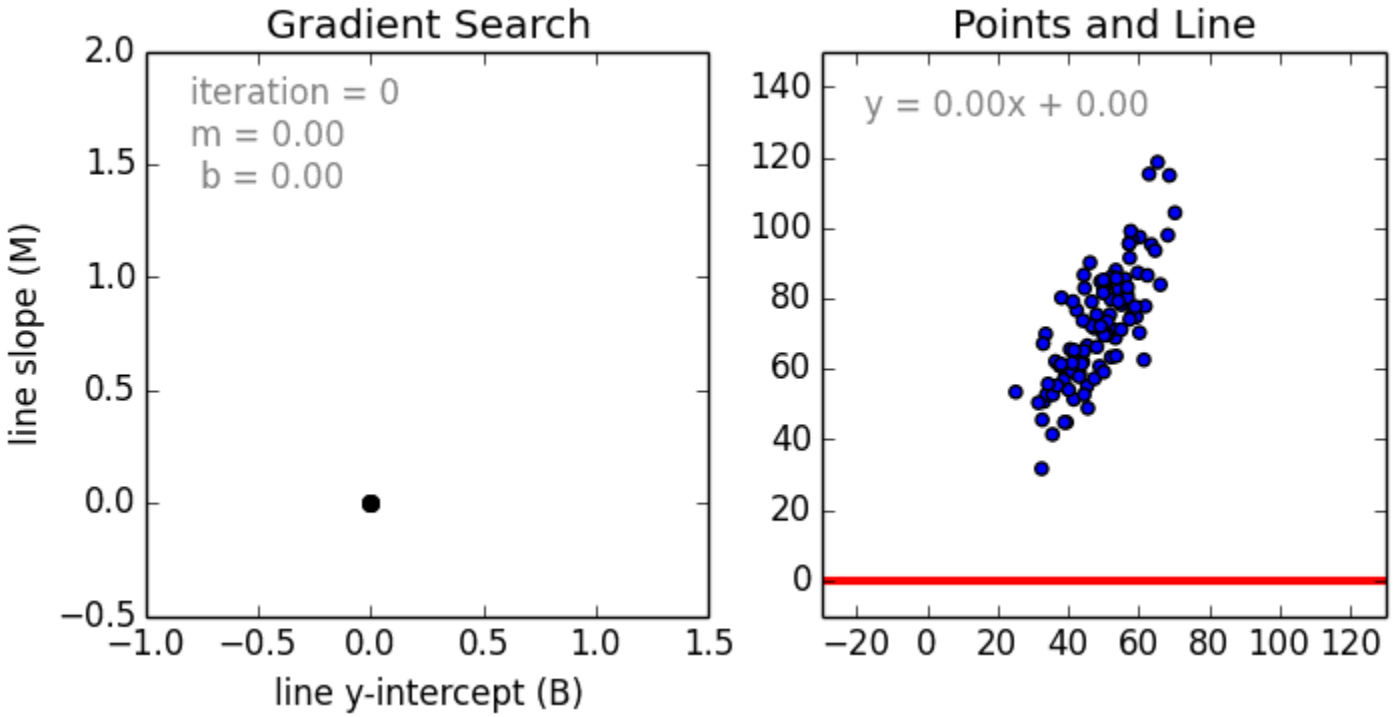
Line: $y = mx + c$

Best line??

Find m & c that form a line with the most minimum error !

Source:
Linear Regression - Gradient Descent - TowardDataScience

Gradient Descent



https://raw.githubusercontent.com/mattnedrich/GradientDescentExample/master/gradient_descent_example.gif

Gradient descent is an optimization algorithm;

→ Find the "best" values of parameters that minimize the **cost function** !

Error = |"Data" – "Model"|

Linear Regression

- **(SSE)= $\varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_n$**

$$E(w,a) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad (2)$$

Note:
y_i : Data - i
$\hat{y}_i$: Model - i

  - Find w and a that minimize this function

- Define the 'Total' error

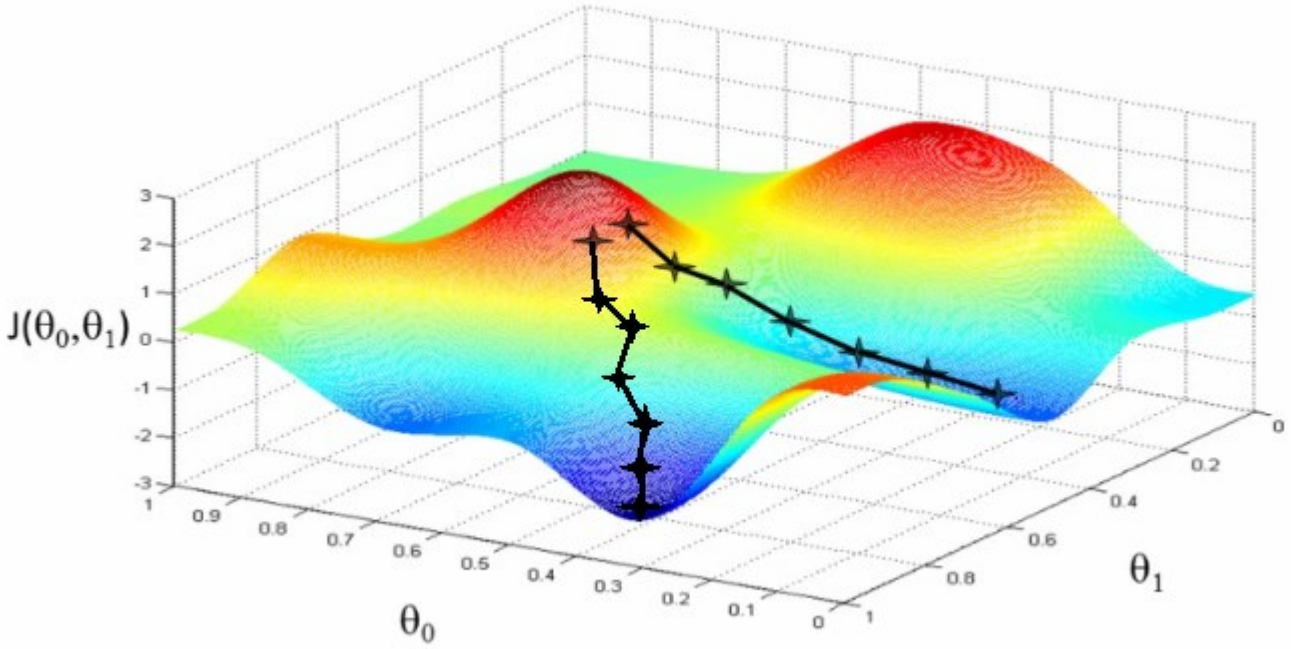
$$E(w,a) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad (2)$$

- or

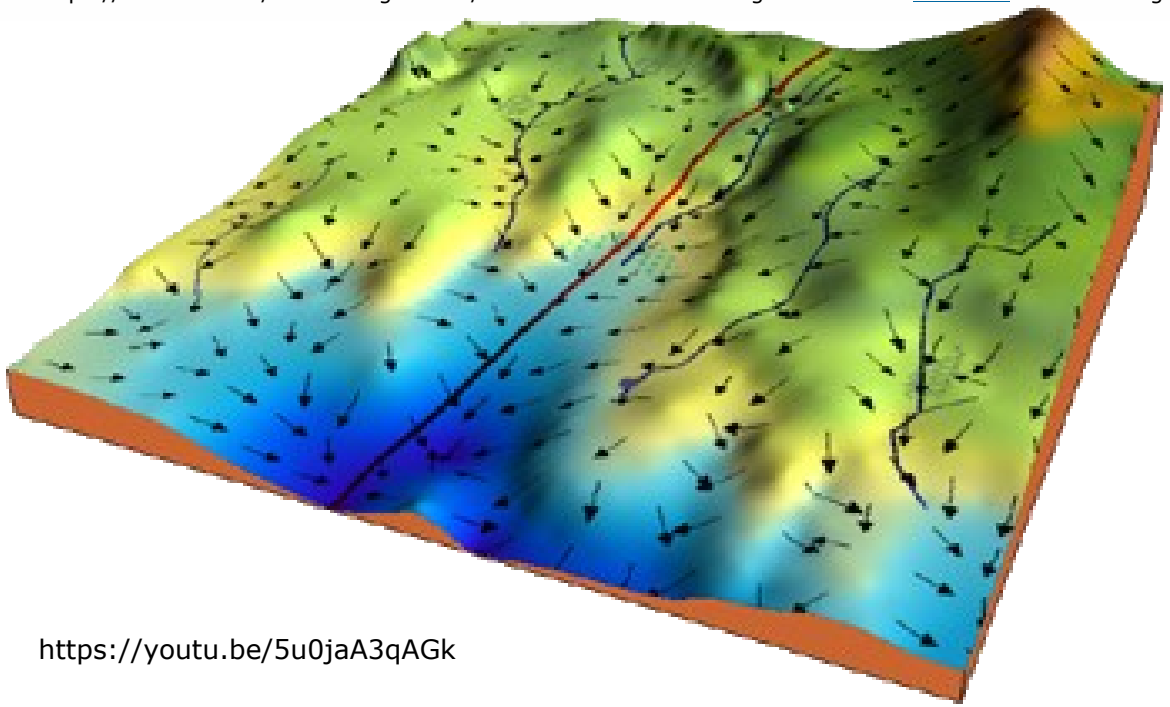
$$E(w,a) = \sum_{i=1}^{n} (y_i - wx_i - a)^2 \qquad (2b)$$

Cost function??

Concept: Gradient Descent



https://medium.com/meta-design-ideas/ Gradient Descent 3D diagram. Source: Coursera — Andrew Ng



https://youtu.be/5u0jaA3qAGk

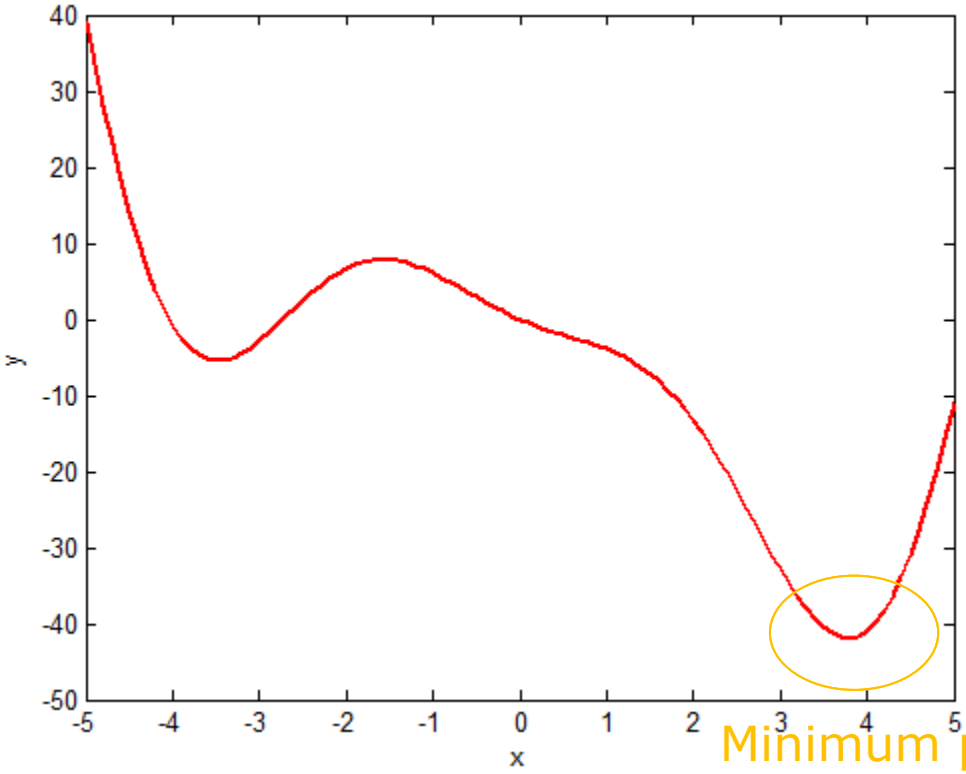
What is the best way (direction) to reach the lowest point in the mountain?

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. *

*) https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

Review:
Minimization of a simple function

Function $f(x) = 2x^2 \cos(x) - 5x$



Minimum point !

Modification from:
https://www.charlesbordet.com/en/gradient-descent/#but-the-maths

Find the minimum of this function $f(x)$!

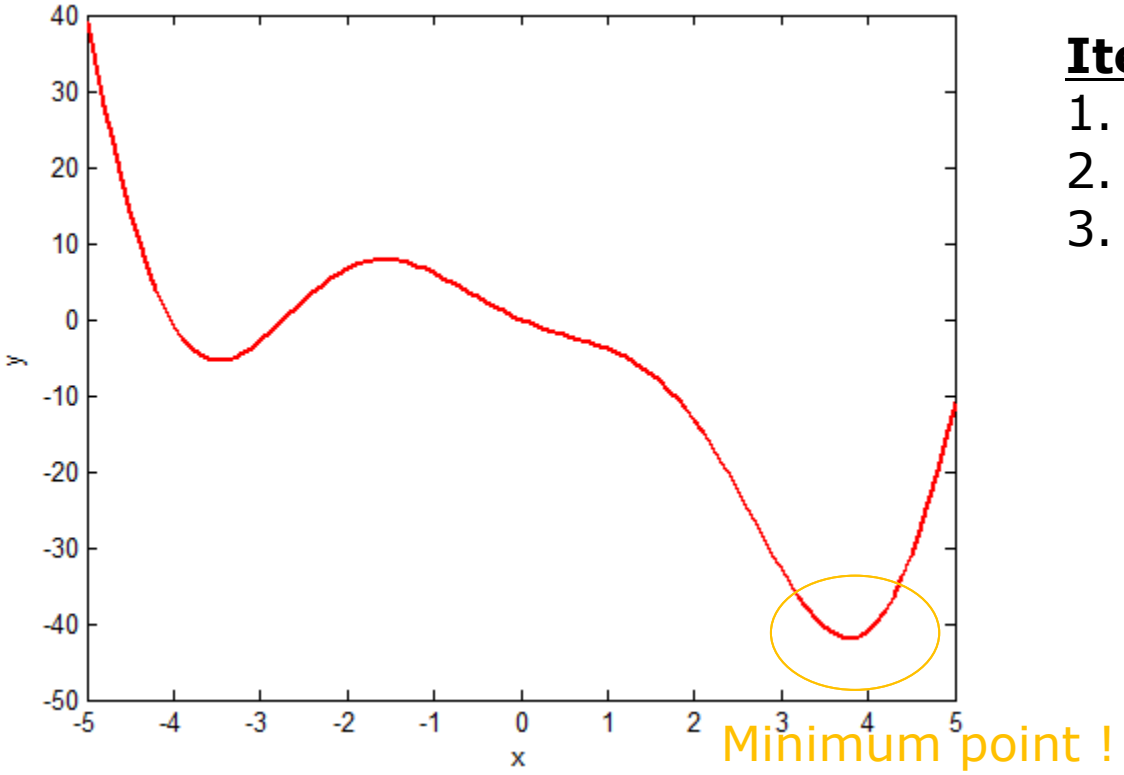
Traditional way:

- Calculate $f'(x)$
- Solve $f'(x_0) = 0$ → x_0 is the critical point !
- x_0 is the minimum or maximum point??
  - Calculate $f''(x)$
  - If $f''(x) > 0$ then x_0 is the minimum point
  - If $f''(x) < 0$ then x_0 is the maximum point

BUT, what is the function is too complex?
Alternative way:

- Solve it iteratively !

Review:
Minimization of a simple function



Modification from:
https://www.charlesbordet.com/en/gradient-descent/#but-the-maths

Iteratively:

1. Take an initial point (guest) x_0
2. Compute $f'(x_0)$
3. In order to find the extreme point;
   - Move the point in the direction opposite to the slope
     $$x_0^{(1)} = x_0^{(0)} - \alpha * f'(x_0^{(0)})$$
   - Iterate this until we find the minimum point !

Note:

- Why negative sign ??
- How big should be the value for α ??
- When should we STOP ??

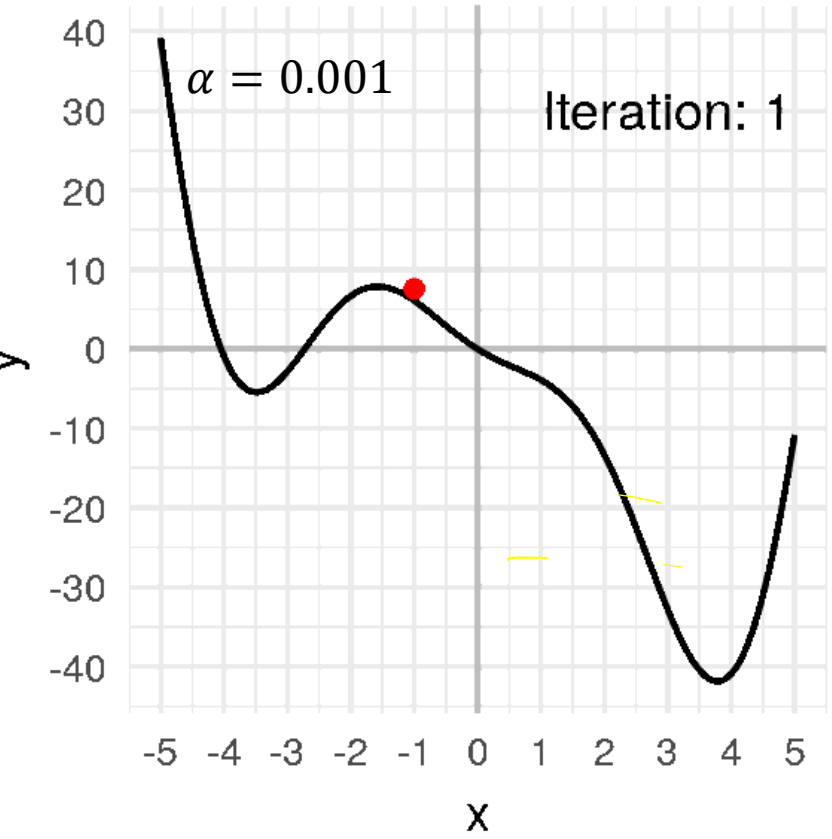
Find the minimum of this function $f(x)$!

Function $f(x) = 2x^2 \cos(x) - 5x$

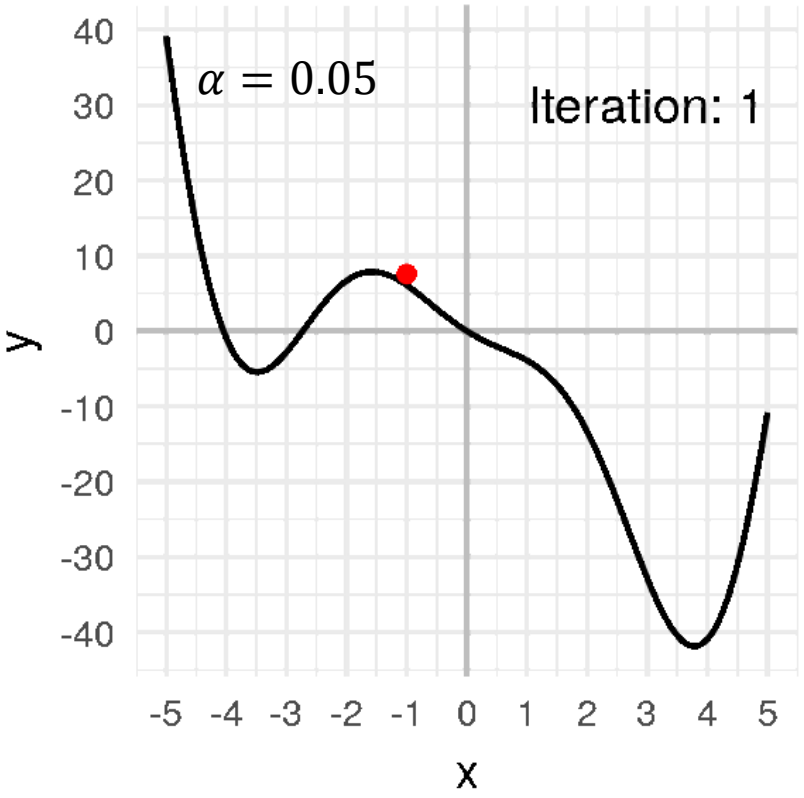
$f'(x) = 4x \cos(x) - \sin(x)2x^2 - 5$

How Sensitive α ??

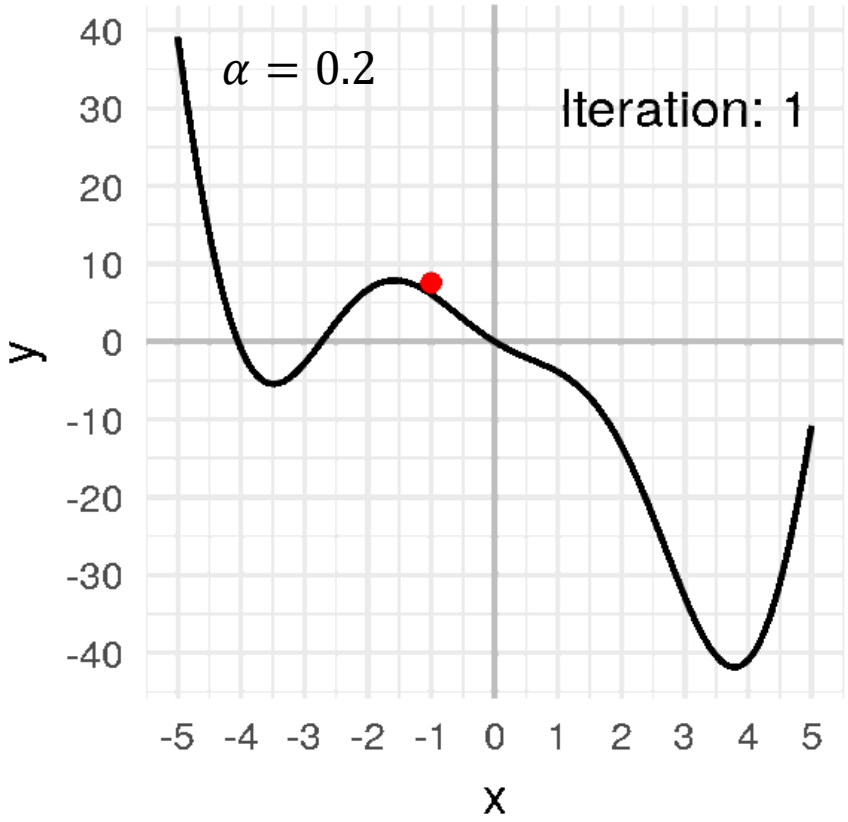
$$x_0^{(1)} = x_0^{(0)} - \alpha * f'(x_0^{(0)})$$



- Many iteration
- Very slow !
- The minimum point is found !



OPTIMUM??

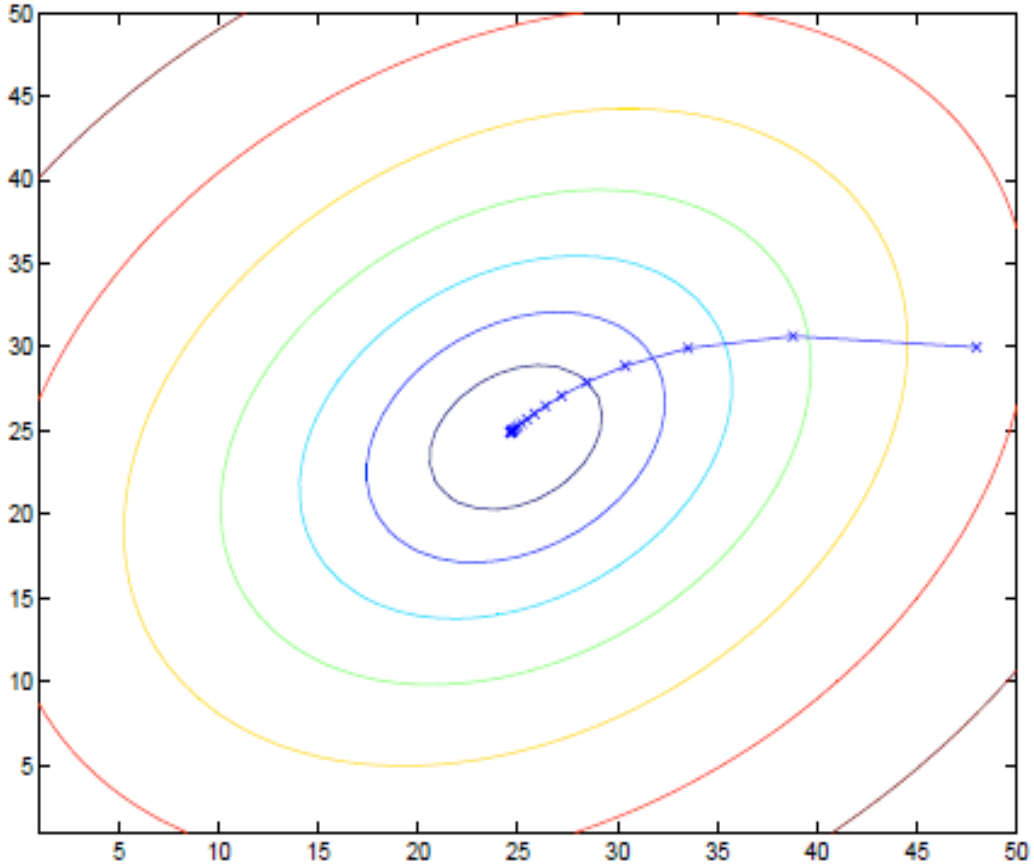


- Very few iteration
- Very fast !
- BUT the minimum point is missed !

Source:
https://www.charlesbordet.com/en/gradient-descent/#but-the-maths

Gradient Descent

Diketahui permukaan $z = f(x,y)$ dengan kurva ketinggian yang dinyatakan pada gambar di sebelah.



Berangkat dari (x_0,y_0), nilai $f(x,y)$ menurun paling cepat, BILA bergerak dalam arah
$-\nabla f(x_0,y_0)$

Contoh:

Bila $f(x,y) = x^2+y^2$ dan $P_0(2,1)$, maka $f(P_0) = 5$.
Arah gerak agar nilai f menurun paling besar di titik P : (-4,-2)

$P_{baru} = P_0 + \alpha(-4,-2) = (2,1) + \alpha(-4,-2)$
Utk $\alpha = 0.1$, $P_{baru} = (1.6,0.8)$. Nilai $f(P_{baru}) = 2.08$
Utk $\alpha = 0.2$, $P_{baru} = (1.2,0.6)$. Nilai $f(P_{baru}) = 1.44+0.36 = 1.8$

Grad. Descent utk Reg. Linear

$$Arg\ \underset{(\theta_1,\theta_0)}{Min}\ J = \sum_{i=1}^{N}(y_i - \theta_1 x_i - \theta_0)^2$$

equivalen dengan

$$Arg\ \underset{(\theta_1,\theta_0)}{Min}\ J = \frac{1}{2}\sum_{i=1}^{N}(y_i - \theta_1 x_i - \theta_0)^2$$

Berangkat dari $(\theta_1^{(0)}, \theta_0^{(0)})$, arah pergerakan yang memberikan penurunan paling besar :

$$-\nabla J\ (\theta_1^{(0)}, \theta_0^{(0)})$$

Sehingga, iterasi

$$(\theta_1^{(k+1)}, \theta_0^{(k+1)}) = (\theta_1^{(k)}, \theta_0^{(k)}) - \alpha \nabla J\ (\theta_1^{(k)}, \theta_0^{(k)})$$

konvergen ke nilai minimum 'lokal' dari J untuk suatu nilai α yang cukup kecil.

Ulangi proses sampai konvergen

$$\theta_1 = \theta_1 + \alpha \sum (y_i - \theta_1 x_i - \theta_0) x_i$$

$$\theta_0 = \theta_0 + \alpha \sum (y_i - \theta_1 x_i - \theta_0)$$

Grad. Descent utk Reg. Linear (Perumuman)

$$Arg\,\underset{\vec{\theta}}{Min}\,J = \frac{1}{2}\sum_{i=1}^{N}(y_i - \theta^T \vec{x}_i)^2$$

Berangkat dari $\vec{\theta}^{(0)}$ arah pergerakan yang memberikan penurunan paling besar :

$$-\nabla J\,(\boldsymbol{\theta}^{(0)})$$

Sehingga, iterasi

$$\theta^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha \nabla J \boldsymbol{\theta}^{(k)},$$

konvergen ke nilai minimum 'lokal'
dari J untuk suatu nilai α yang cukup kecil.

Ulangi proses sampai konvergen

$\theta_k = \theta_k + \alpha\sum(y_i - \theta^T \mathbf{x}_i)x_{ik,}$ k=1,2,…,M

$X_{i0} = 1$ untuk semua i=1,2,…,N

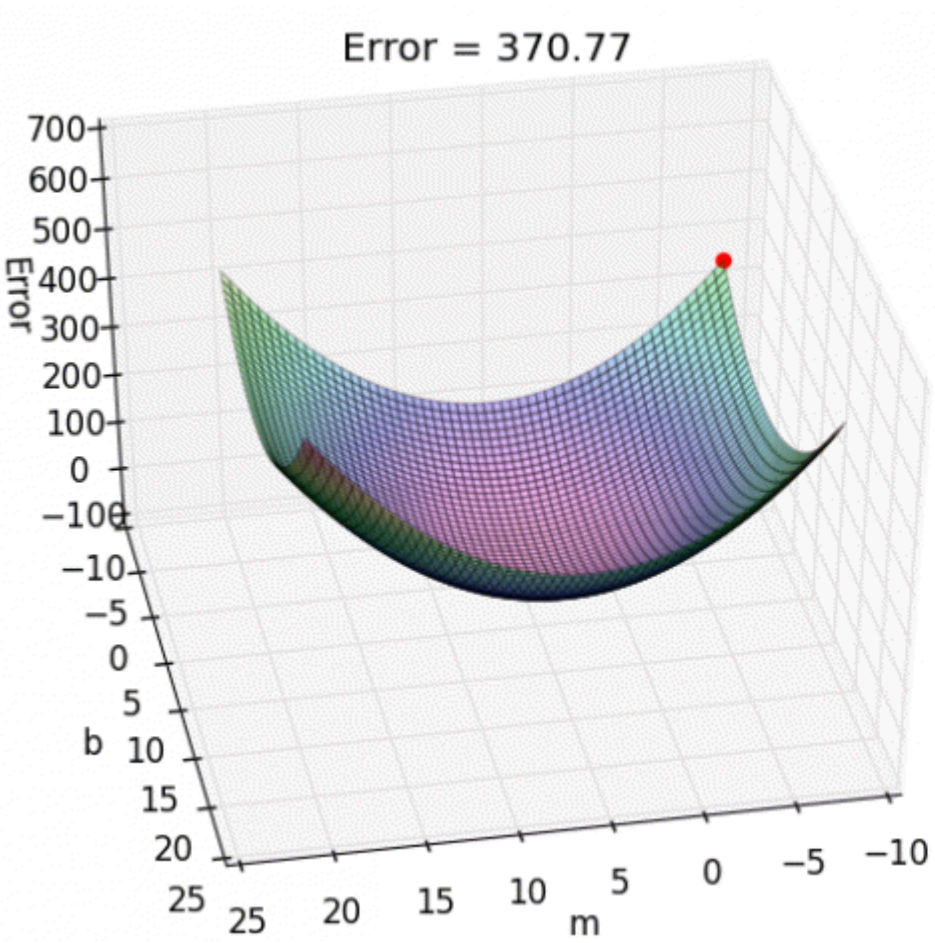
Gradient Descent for Linear Regression

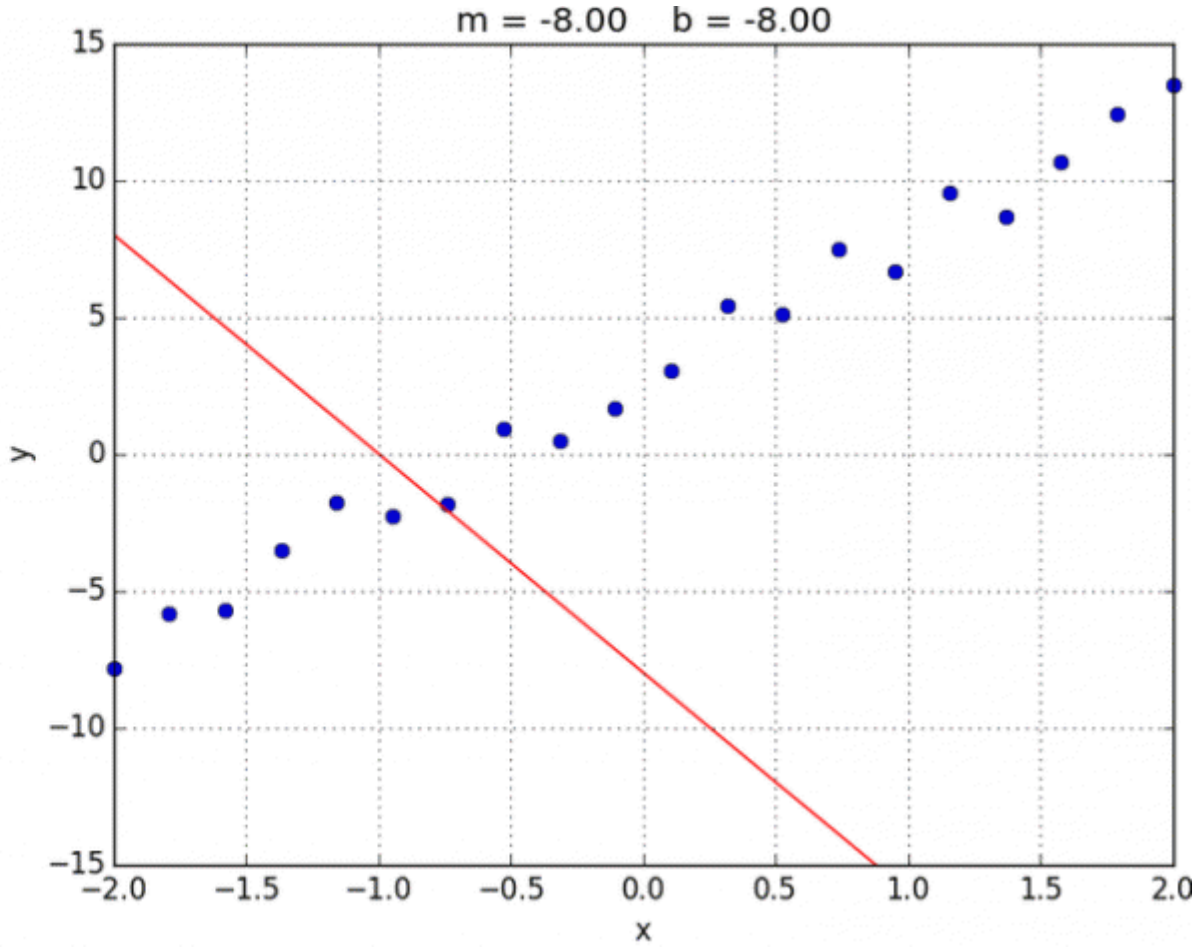
Line:

$y = wx + a$

$y = mx + c$

Minimize the Error !! $E(w, a)$ or $E(m, b)$





Source: https://alykhantejani.github.io/images/gradient_descent_line_graph.gif

Variants of Gradient Descent

Gradient descent is the preferred way to optimize neural networks and many other machine learning algorithms but is often used as **a black box**. … (https://ruder.io/optimizing-gradient-descent/)

Gradient descent is one of the most popular algorithms to **perform optimization** and by far the most common way to **optimize neural networks**. At the same time, every state-of-the-art Deep Learning library contains implementations of various algorithms to optimize gradient descent (e.g. lasagne's, caffe's, and keras' documentation). …

Gradient descent variants

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

Source:

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Variants of Gradient Descent

Gradient descent is the preferred way to optimize neural networks and many other machine learning algorithms but is often used as **a black box**. … (https://ruder.io/optimizing-gradient-descent/)

Gradient descent is one of the most popular algorithms to **perform optimization** and by far the most common way to **optimize neural networks**. At the same time, every state-of-the-art Deep Learning library contains implementations of various algorithms to optimize gradient descent (e.g. lasagne's, caffe's, and keras' documentation). …

Gradient descent variants

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

Source:

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

1. Batch gradient descent

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

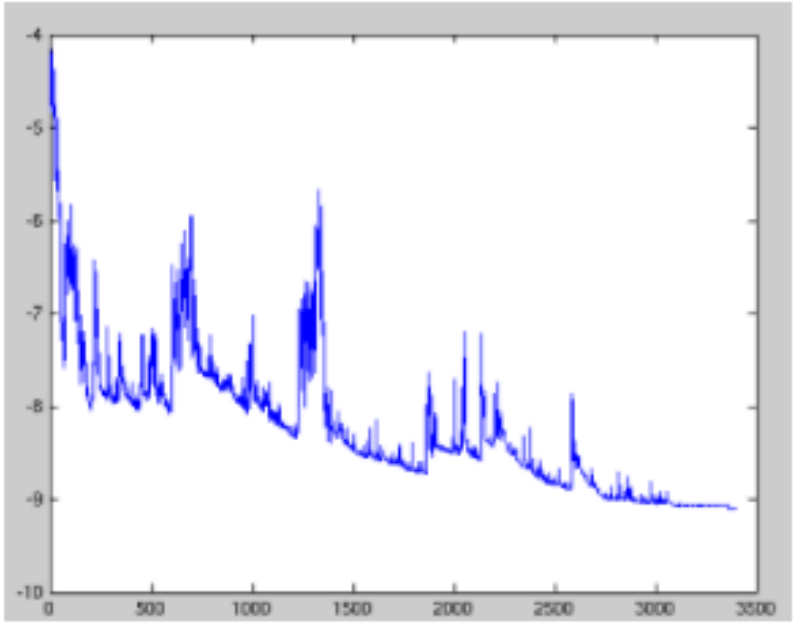
- computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset
- As we need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be **very slow** and is intractable for datasets that don't fit in memory. Batch gradient descent also doesn't allow us to update our model online, i.e. with new examples on-the-fly.

Source:
Ruder, S. (2016)

2. Stochastic gradient descent

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

- Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x^i and y^i
- Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.
- SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily as in Image 1.



Source:
Ruder, S. (2016)

3. Mini-batch gradient descent

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}).$$

- Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples
- This way, it
  - reduces the variance of the parameter updates, which can lead to more stable convergence; and
  - can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.
- Common mini-batch sizes range between 50 and 256, but can vary for different applications. Mini-batch gradient descent is typically the algorithm of choice when training a neural network and the term SGD usually is employed also when mini-batches are used. Note: In modifications of SGD in the rest of this post, we leave out the parameters $\mathrm{x}^{(i;i+1)}$ and $y^{(i;i+1)}$ for simplicity.

Source:
Ruder, S. (2016)

Gradient descent optimization algorithms

- Momentum
- Nesterov accelerated gradient
- Adagrad
- Adadelta
- RMSprop
- Adam
- AdaMax
- Nadam
- AMSGrad

Source:
Ruder, S. (2016)

4.1 Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [20], which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum as in Figure 2a.

(a) SGD without momentum

(b) SGD with momentum

Figure 2: Source: Genevieve B. Orr

Momentum [17] is a method that helps accelerate SGD in the relevant direction and dampens oscillations as can be seen in Figure 2b. It does this by adding a fraction γ of the update vector of the past time step to the current update vector[8]:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{4}$$

The momentum term γ is usually set to 0.9 or a similar value.

Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity, if there is air resistance, i.e. $\gamma < 1$). The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

Source: Ruder, S. (2016)

4.6 Adam

Adaptive Moment Estimation (Adam) [10] is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{19}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).

They counteract these biases by computing bias-corrected first and second moment estimates:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{20}$$

They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{21}$$

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

Source: Ruder, S. (2016)