Machine learning

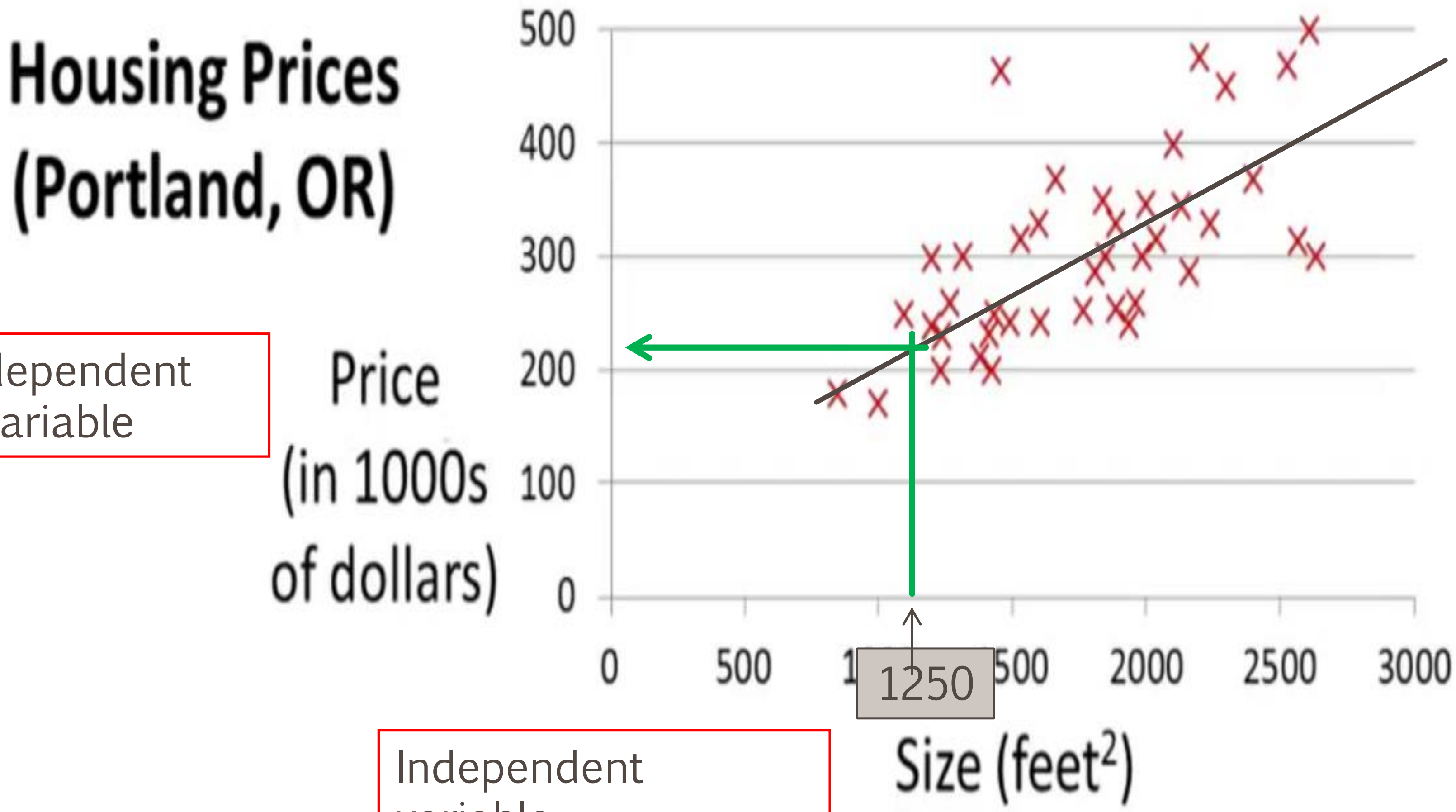
Presented by : Dr. Hanaa Bayomi

Lecture 2 : Linear Regression

LINEAR REGRESSION WITH ONE VARIABLE

- Model Representation
- Cost Function
- Gradient Descent

MODEL REPRESENTATION



dependent variable

Independent variable

Supervised Learning

"right answers" or "Labeled data" given

Regression:

Predict continuous valued output (price)

MODEL REPRESENTATION

Training set of housing prices (Portland, OR)

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

m

Notation:

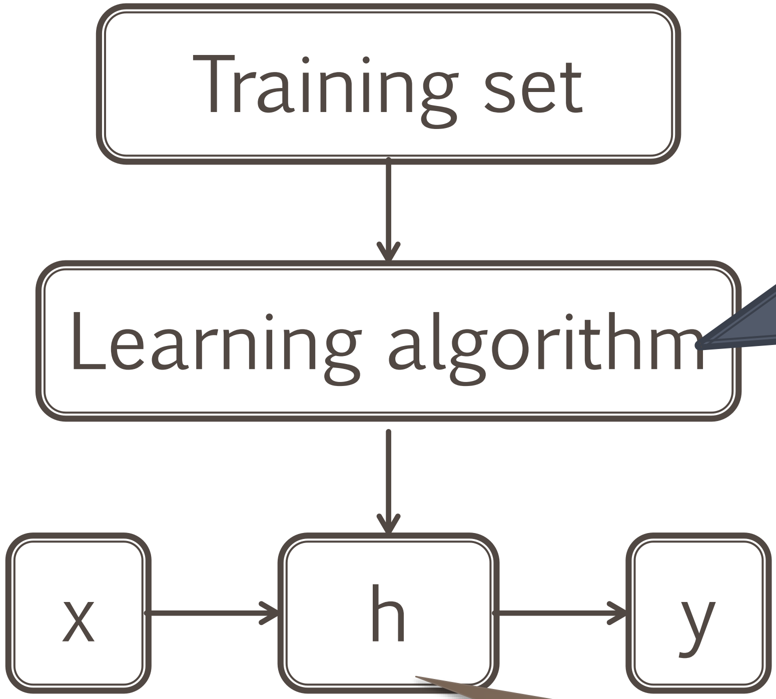
m = Number of training examples
x's = "input" variable / features
y's = "output" variable / "target" variable

(x,y) **one training example (one raw)**
$(x^{(i)}, y^{(i)})$ i th training example

Example

$x^{(1)}$	2104
$y^{(2)}$	232
$x^{(4)}$	852

MODEL REPRESENTATION



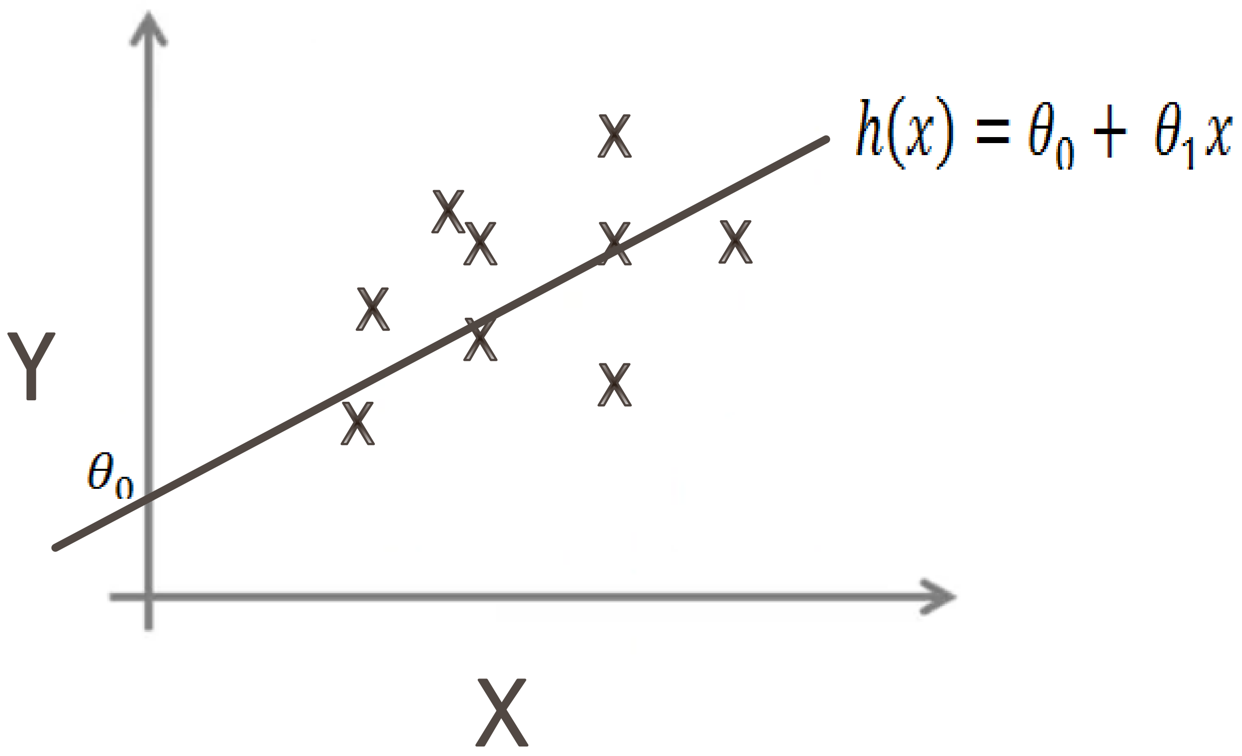
the job of a learning algorithm to output a function is usually denoted lowercase **h** and **h** stands for hypothesis

the job of a hypothesis function is taking the value of x and it tries to output the estimated value of y. So h is a function that maps from x's to y's

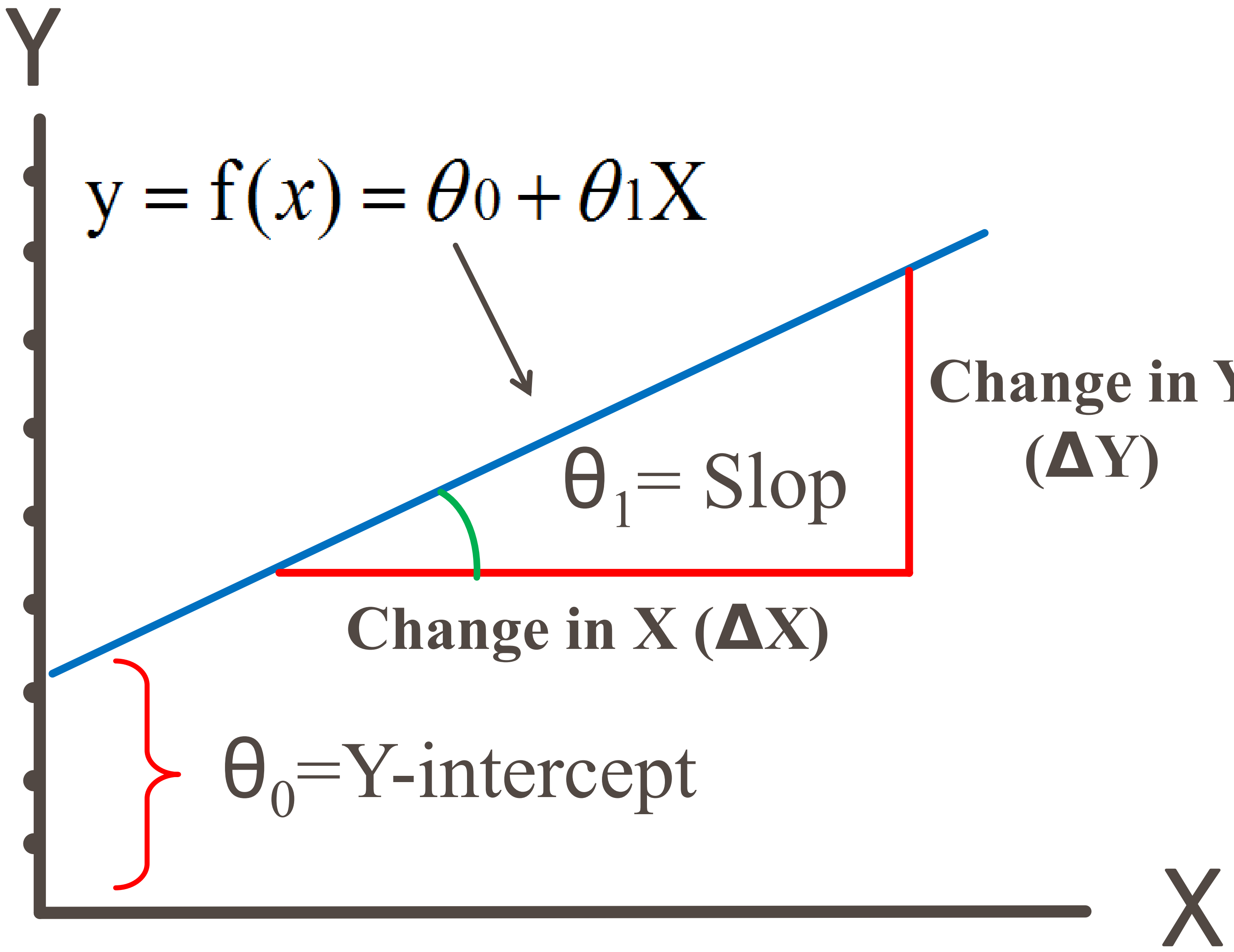
MODEL REPRESENTATION

How do we represent h ?

$$h(x) = \theta_0 + \theta_1 x$$

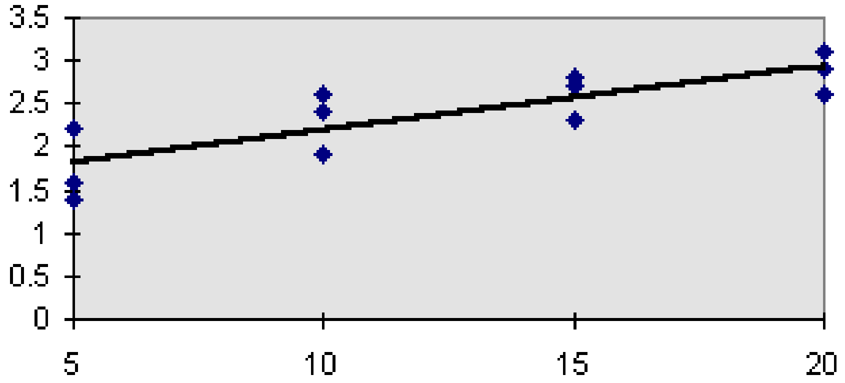


Linear Equations

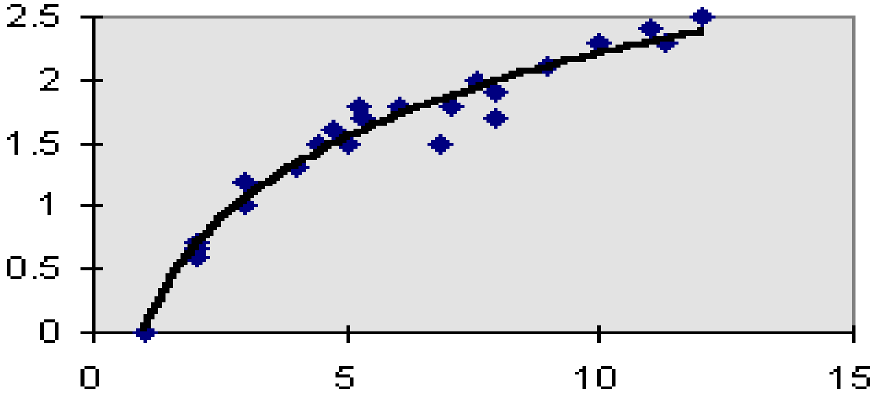


Types of Regression Models

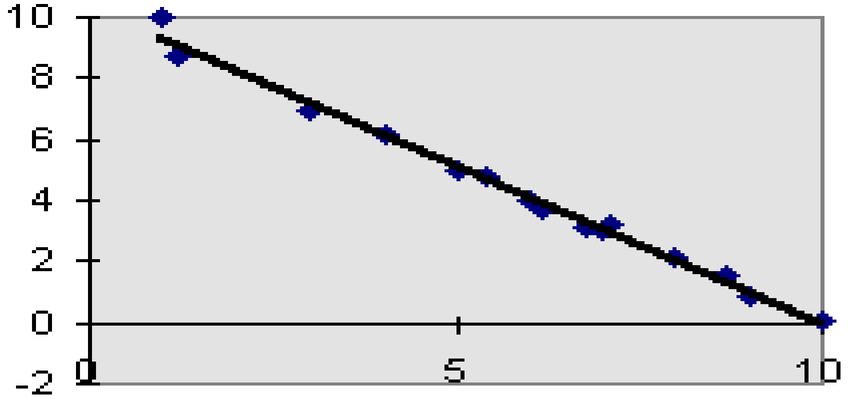
Positive Linear Relationship



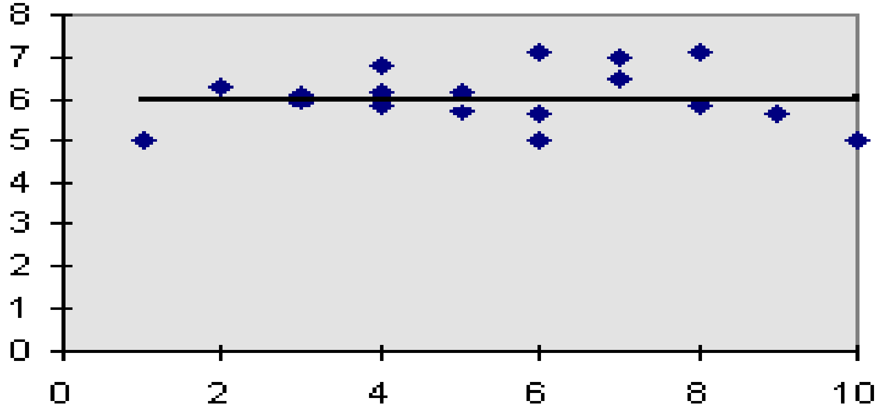
Relationship NOT Linear



Negative Linear Relationship



No Relationship



COST FUNCTION

- *The cost function*, let us figure out how to fit the best possible straight line to our data.

Training Set

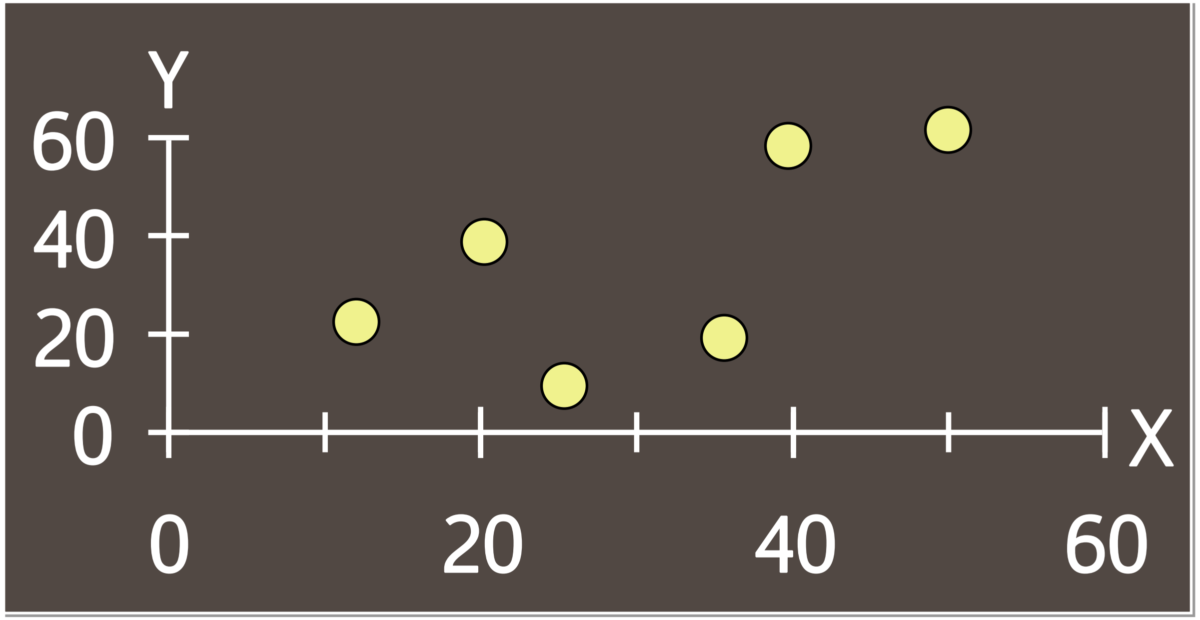
Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

How to choose $\theta_{i's}$?

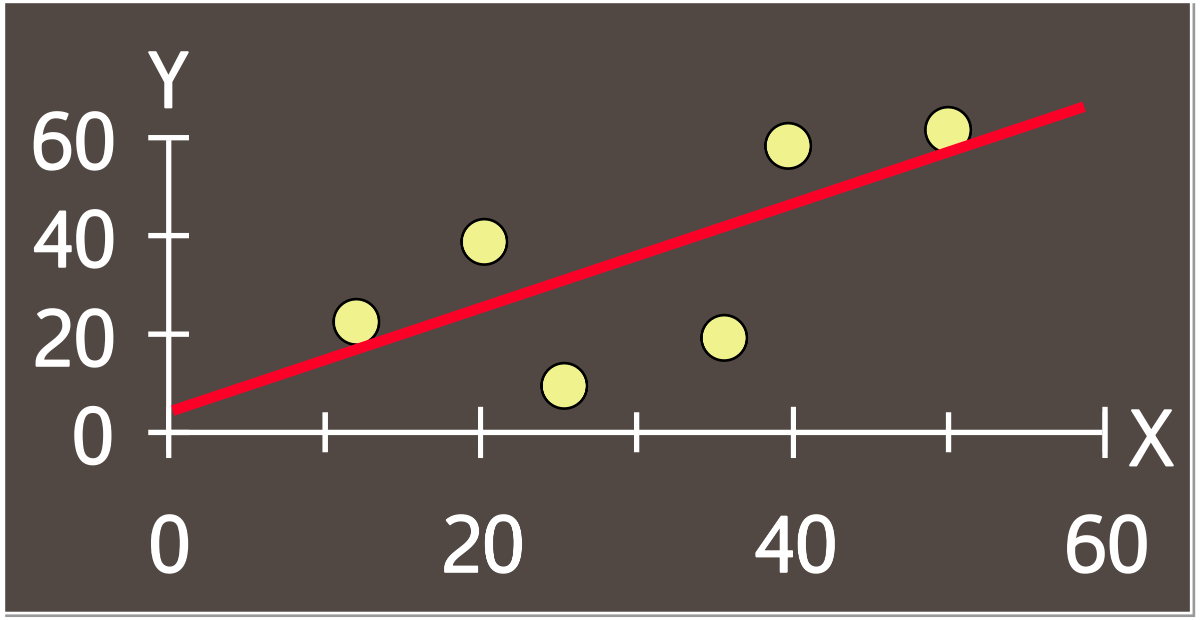
Scatter plot

- 1. Plot of All (X_i, Y_i) Pairs
- 2. Suggests How Well Model Will Fit



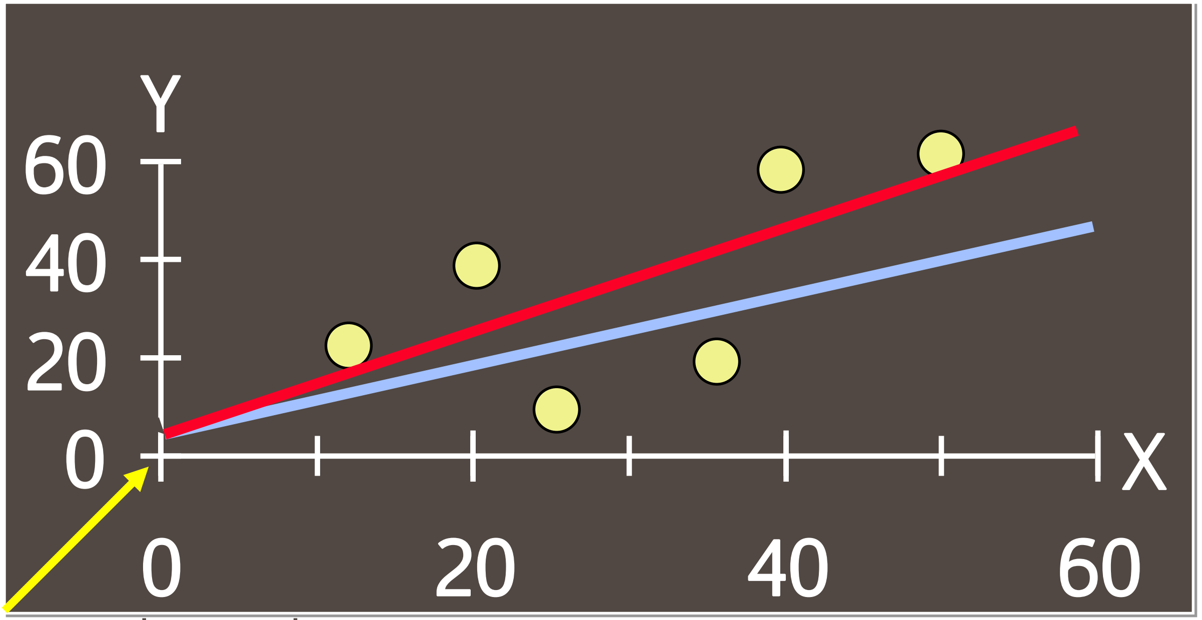
Thinking Challenge

How would you draw a line through the points?
How do you determine which line 'fits best'?



Thinking Challenge

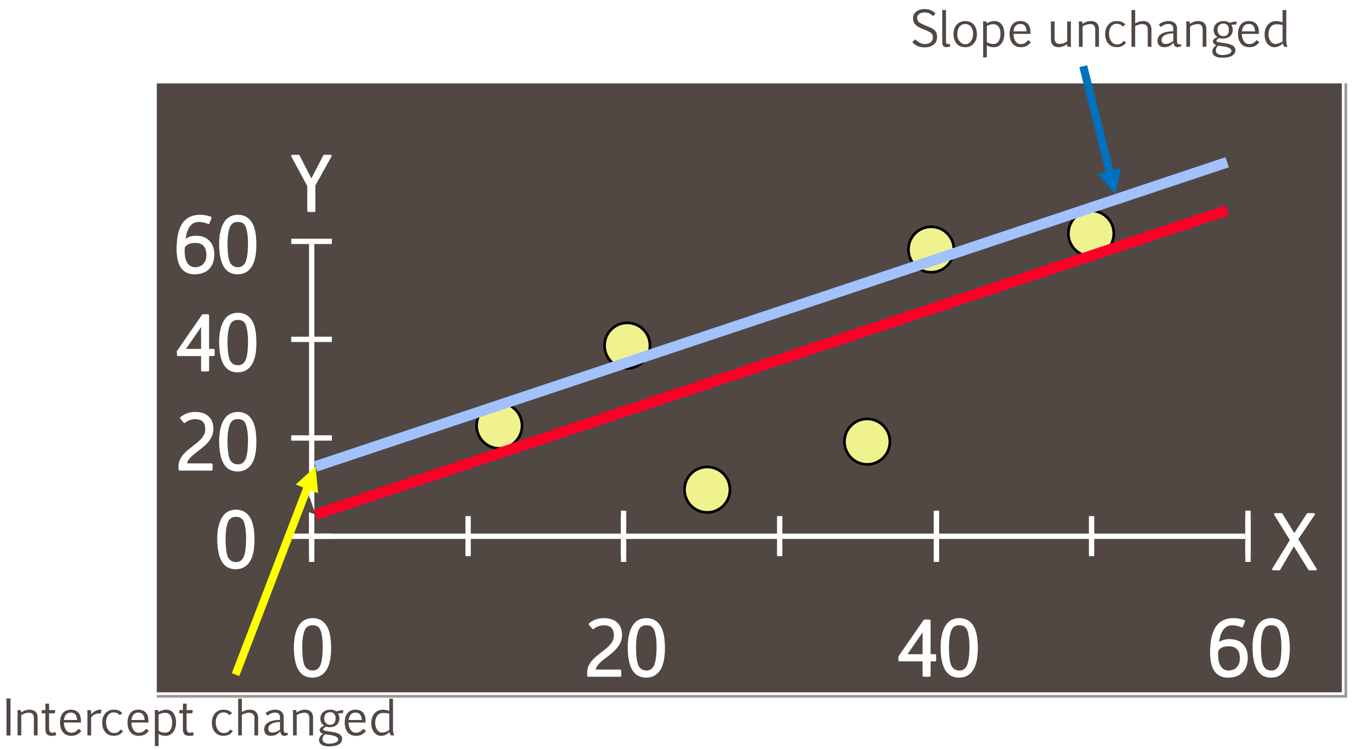
How would you draw a line through the points?
How do you determine which line ‘fits best’?



Intercept unchanged

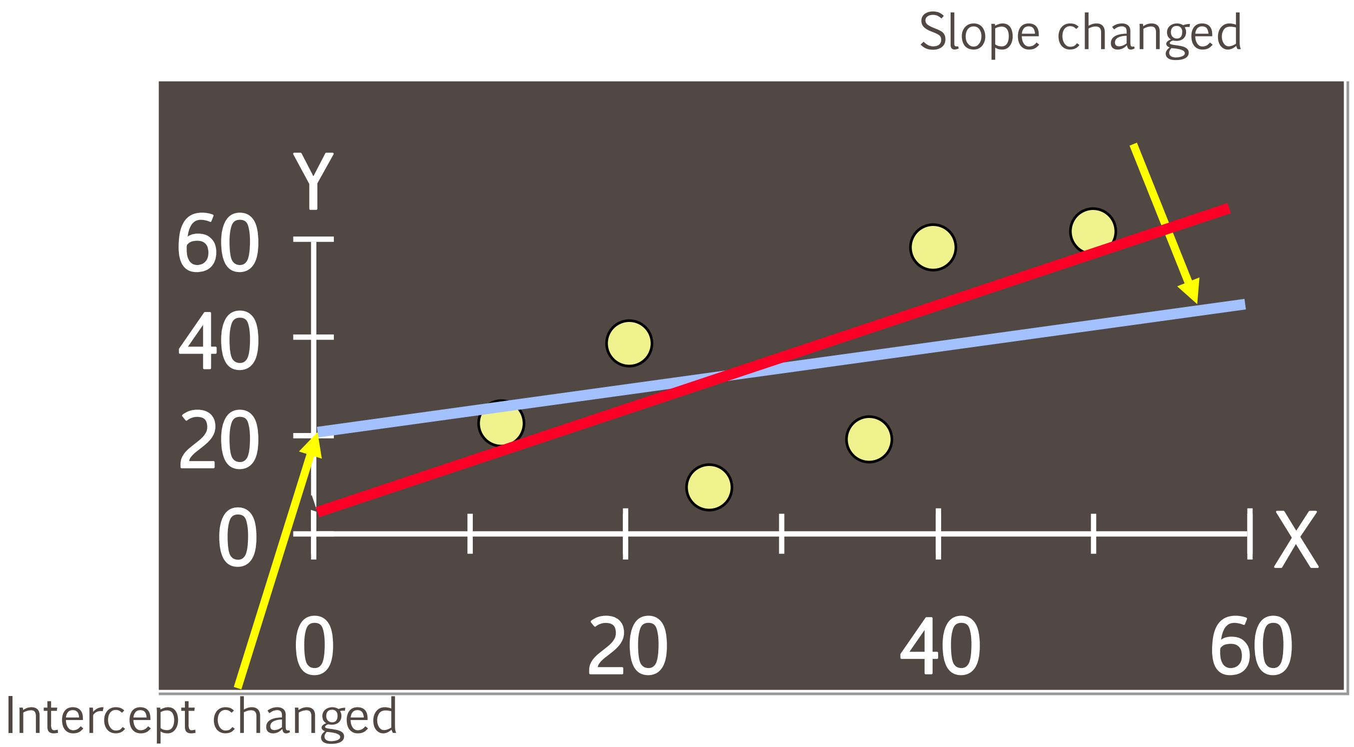
Thinking Challenge

How would you draw a line through the points?
How do you determine which line 'fits best'?



Thinking Challenge

How would you draw a line through the points?
How do you determine which line 'fits best'?



Least Squares

- 1. 'Best Fit' Means Difference Between Actual Y Values and Predicted Y Values is a Minimum. So square errors!

$$\sum_{i=1}^{m}\left(Y_i - h\theta(x_i)\right)^2 = \sum_{i=1}^{m}\hat{\varepsilon}_i^2$$

Least Squares

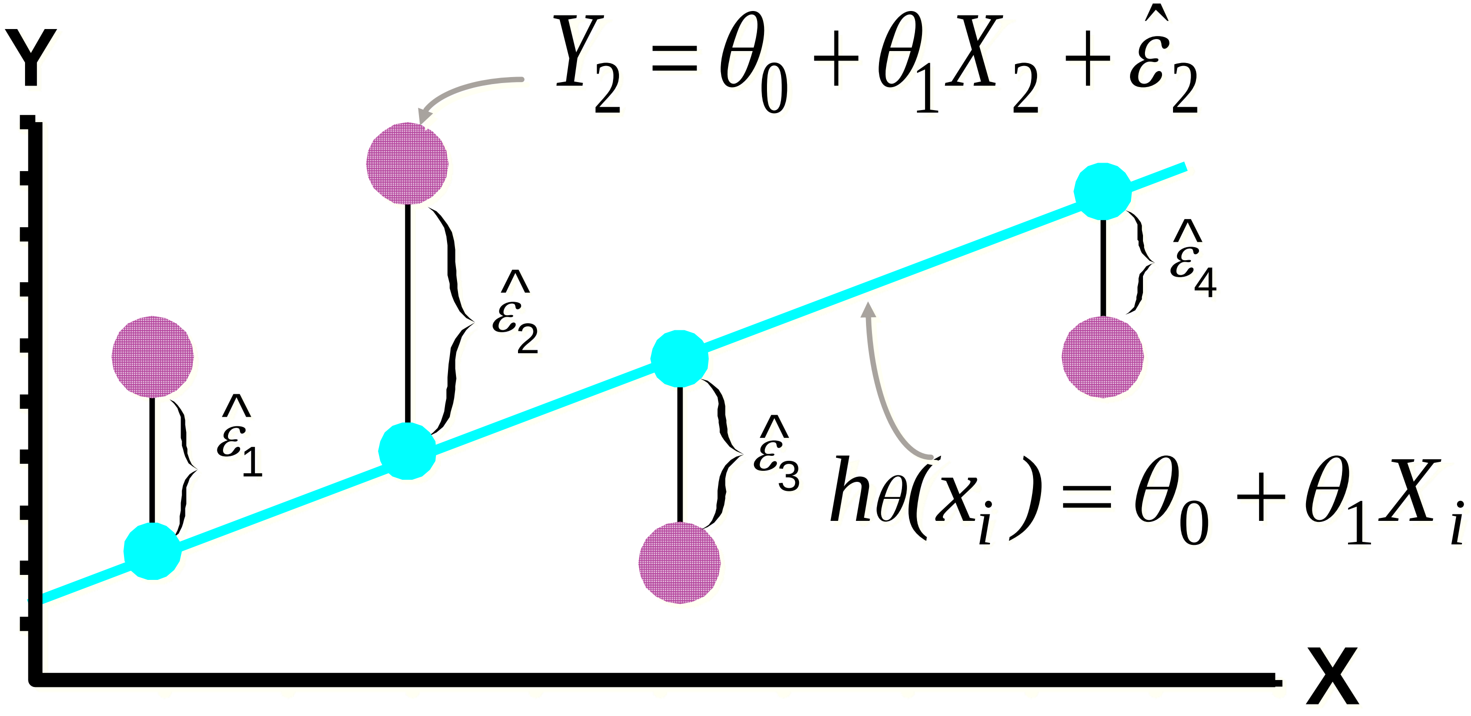
- 1. 'Best Fit' Means Difference Between Actual Y Values & Predicted Y Values Are a Minimum. So square errors!

$$\sum_{i=1}^{m}\left(Y_i - h\theta(x_i)\right)^2 = \sum_{i=1}^{m}\hat{\varepsilon}_i^2$$

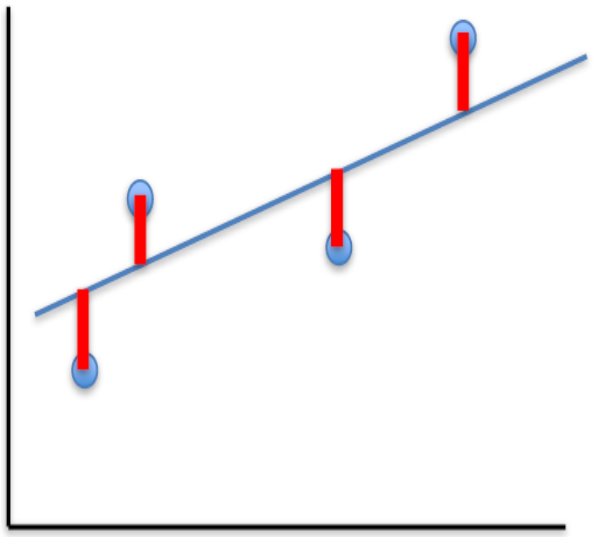
- 2. LS Minimizes the Sum of the Squared Differences (errors) (SSE)

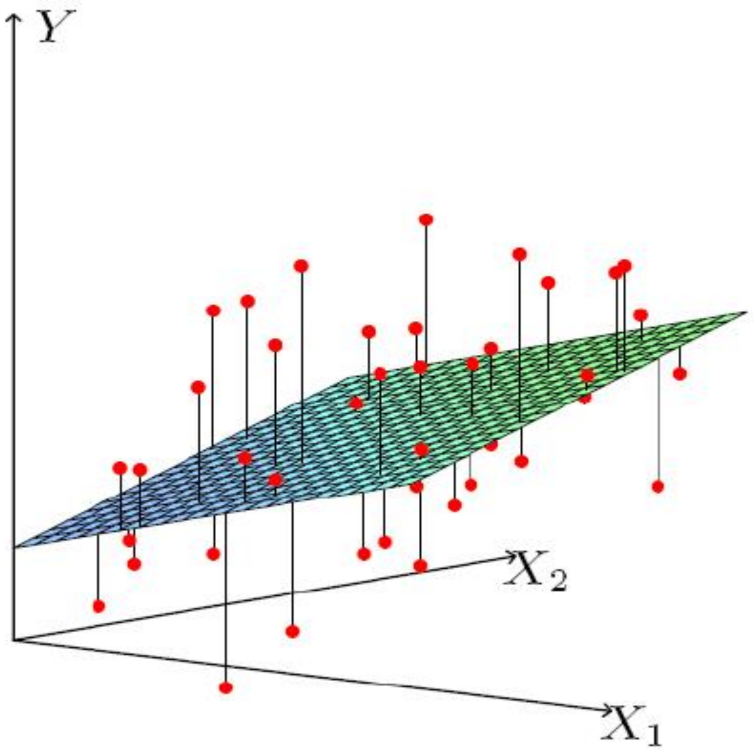
Least Squares Graphically

$$\text{LS minimizes } \sum_{i=1}^{n} \hat{\varepsilon}_i^2 = \hat{\varepsilon}_1^2 + \hat{\varepsilon}_2^2 + \hat{\varepsilon}_3^2 + \hat{\varepsilon}_4^2$$



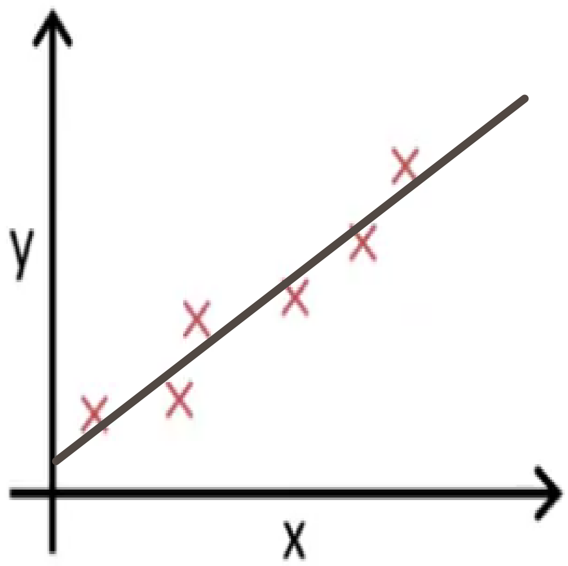
Least Squared **errors** Linear Regression





$$minimize_{\theta_0,\theta_1} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

COST FUNCTION



Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

Minimize $\theta_0 \;\; \theta_1$ $$\frac{1}{2m}\sum_{i}^{m}(h_\theta(x^i) - y^i)^2$$

$$h_\theta(x^i) = \theta_0 + \theta_1 x^i$$

$h_\theta(x^i)$ predictions on the training set

y^i the actual values

$$j(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i}^{m}(h_\theta(x^i) - y^i)^2$$

Minimize $\theta_0 \;\; \theta_1$ $j(\theta_0, \theta_1)$

Cost function visualization

Consider a simple case of hypothesis by setting θ_0=0, then ***h*** becomes : **$h_\theta(x)=\theta_1 x$**

Each value of θ_1 corresponds to a different hypothesis as it is the **slope** of the line

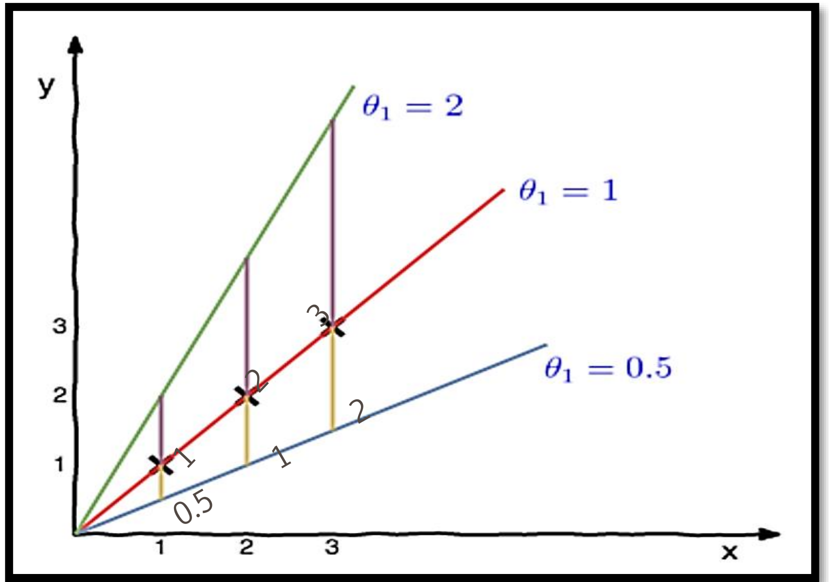
which corresponds to different lines passing through the **origin** as shown in plots below as **y-intercept** i.e. θ_0 is nulled out.

$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(\theta_1 x^{(i)} - y^{(i)}\right)^2$$

At θ_1=2, $J(2) = \frac{1}{2*3}(1^2+2^2+3^2) = \frac{14}{6} = 2.33$

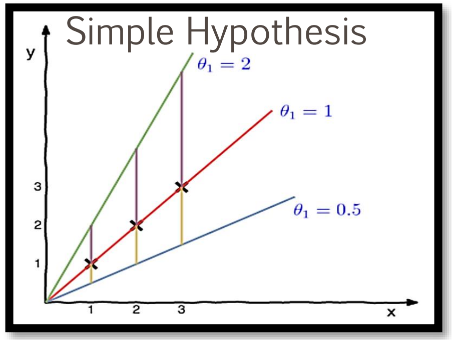
At θ_1=1, $J(1) = \frac{1}{2*3}(0^2+0^2+0^2) = 0$

At θ_1=0.5, $J(= \frac{1}{2*3}(0.5^2+1^2+1.5^2) = 0.58$



Cost function visualization

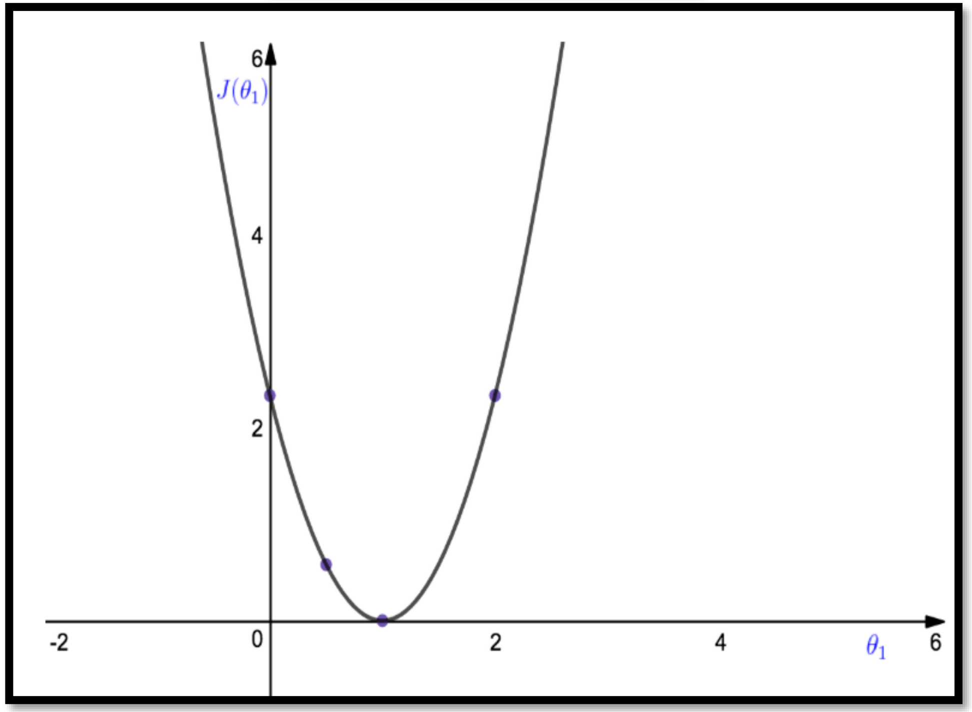
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_1 x^{(i)} - y^{(i)}\right)^2$$



At θ_1=2, $J(2) = \frac{1}{2*3}(1^2 + 2^2 + 3^2) = \frac{14}{6} = 2.33$

At θ_1=1, $J(1) = \frac{1}{2*3}(0^2 + 0^2 + 0^2) = 0$

At θ_1=0.5, $J(0.5) = \frac{1}{2*3}(0.5^2 + 1^2 + 1.5^2) = 0.58$



On **plotting points** like this further, one gets the following graph for the cost function which is dependent on parameter θ_1.

plot each value of θ_1 corresponds to a different hypothesizes

Cost function visualization

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_1 x^{(i)} - y^{(i)} \right)^2$$

What is the optimal value of **θ_1 that minimizes $J(\theta_1)$?**

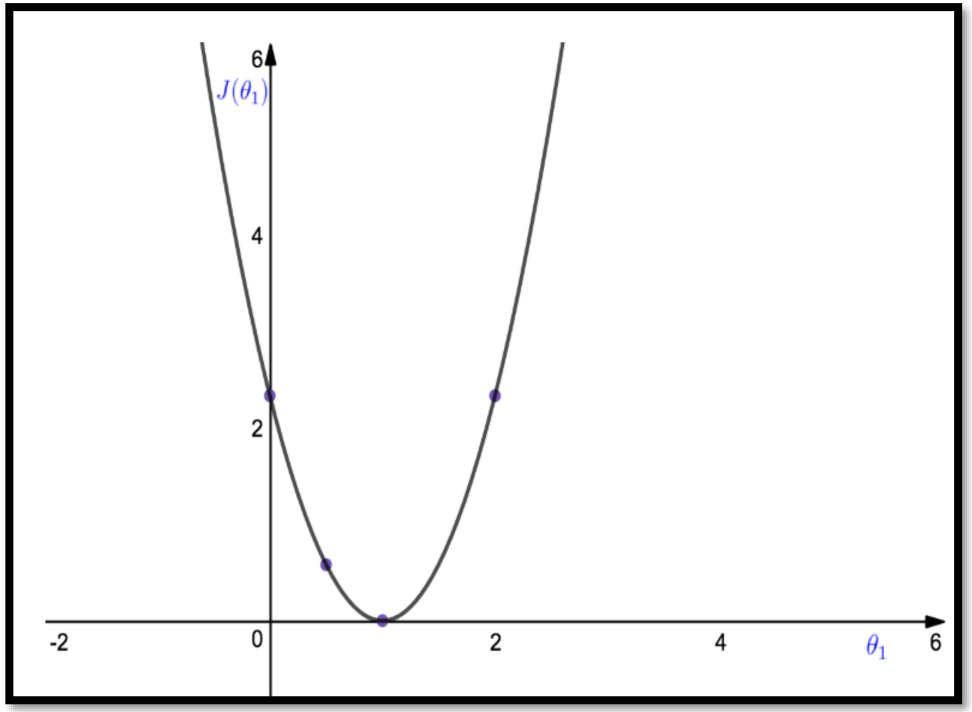
It is clear that best value for θ_1 =1 as $J(\theta_1) = 0$, which is the minimum.

How to find the best value for θ_1 ?

Plotting ?? Not practical specially in high dimensions?

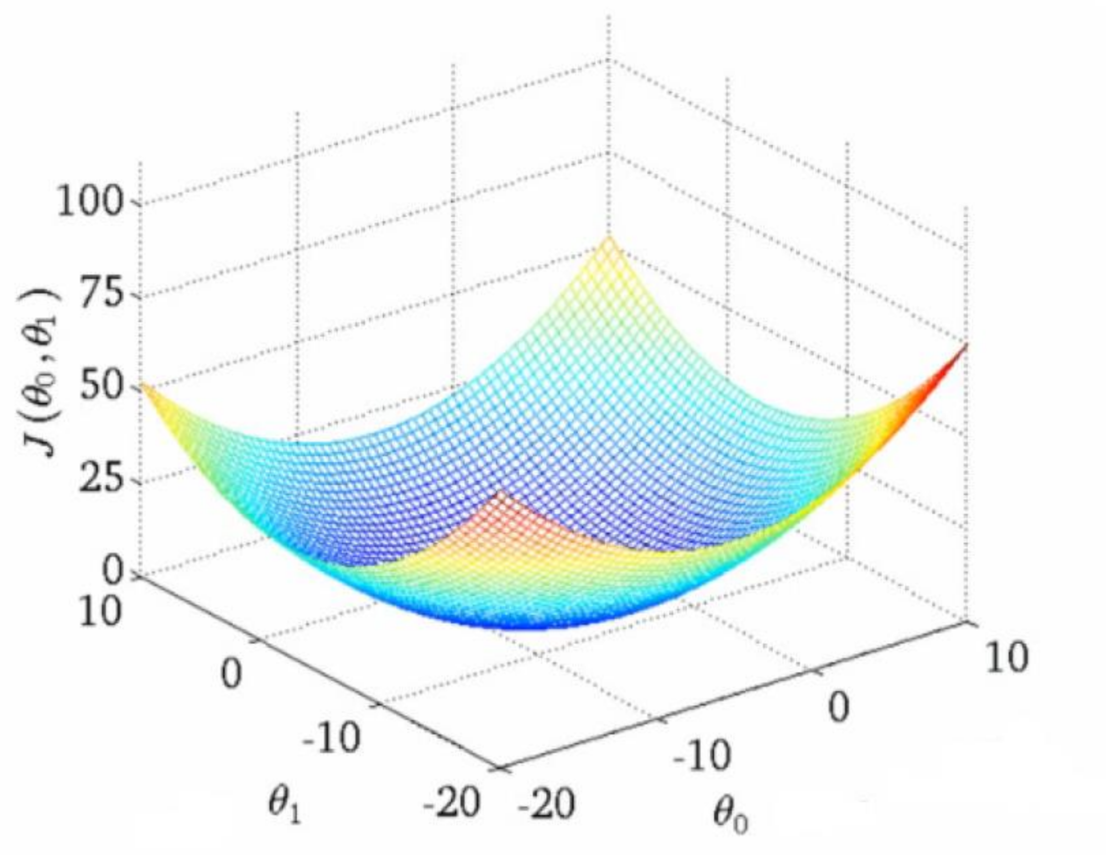
The solution :

1. **Analytical solution: not applicable for large datasets**
2. **Numerical solution: ex: Gradient descent .**



ploting the cost function $j(\theta_0, \theta_1)$

- Previously we plotted our cost function by plotting
  - θ_1 vs $J(\theta_1)$
- Now we have two parameters
  - Plot becomes a bit more complicated
  - Generates a 3D surface plot where axis are
    - $X = \theta_1$
    - $Z = \theta_0$
    - $Y = J(\theta_0, \theta_1)$



COST FUNCTION (RECAP)

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Gradient Descent

GRADIENT DESCENT

- Iterative solution not only in linear regression. It's actually used all over the place in machine learning.

- Objective: minimize any function (Cost Function J)

PROBLEM SETUP

Have some function $J(\theta_0, \theta_1)$

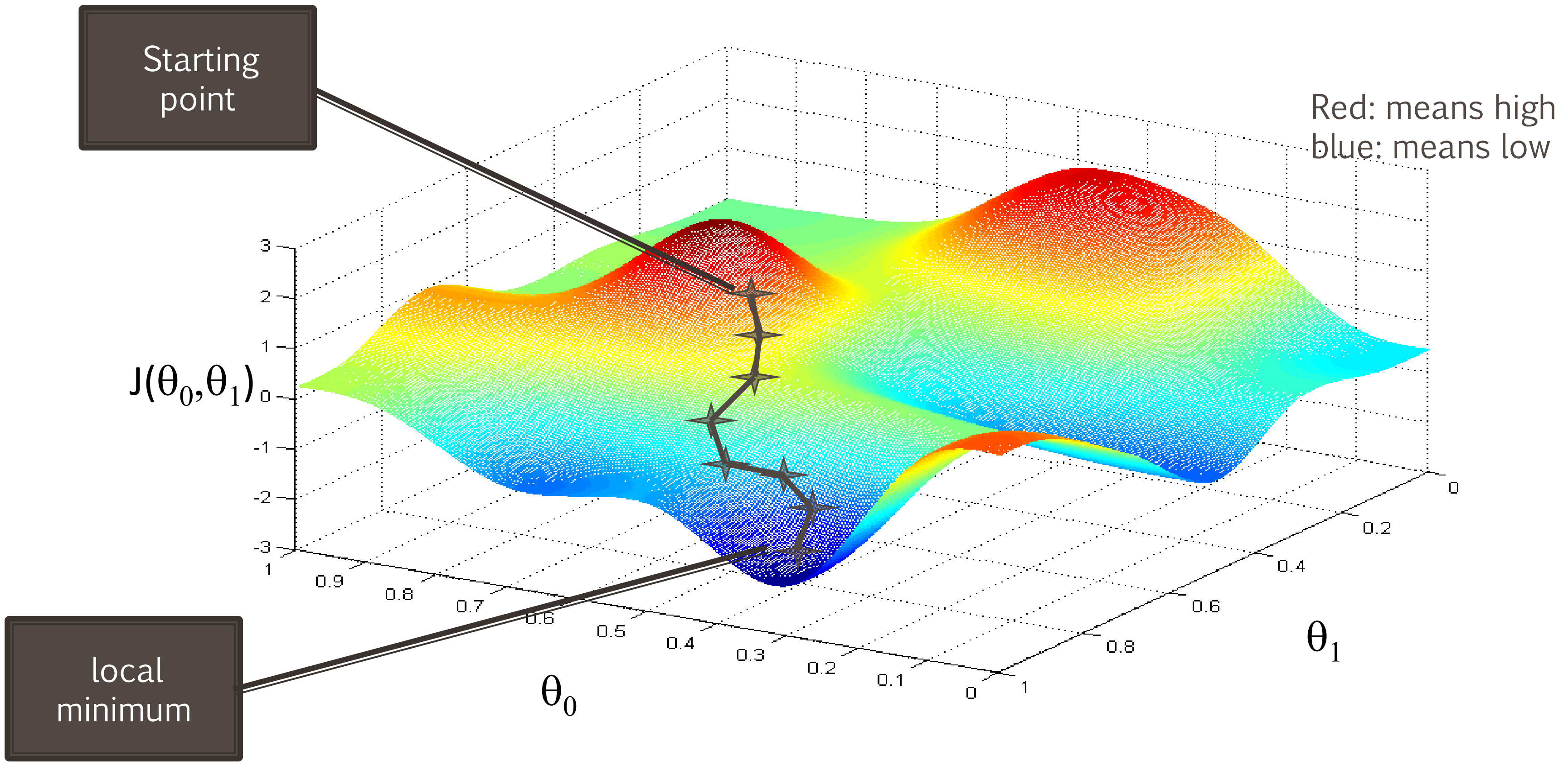
Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

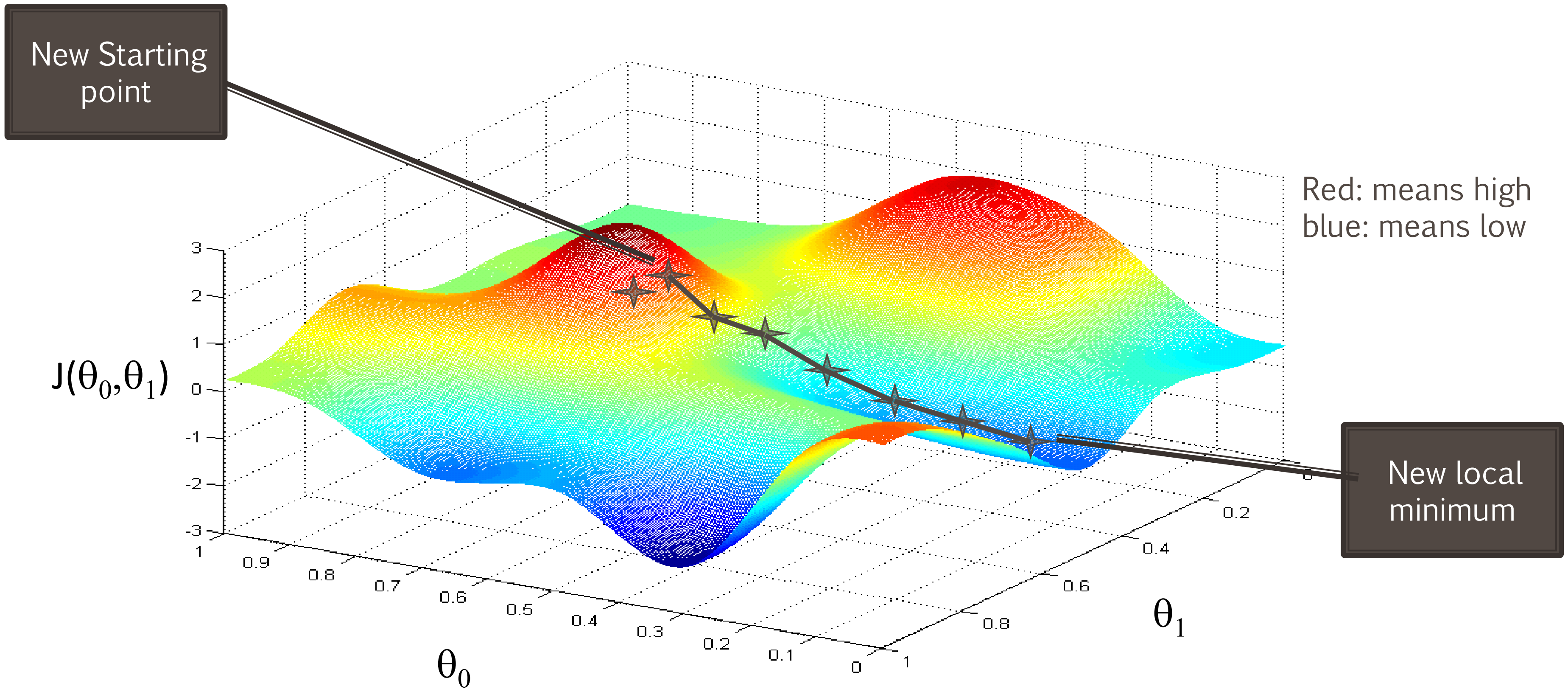
Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

  until we hopefully end up at a minimum

Imagine that this is a landscape of grassy park, and you want to go to the lowest point in the park as rapidly as possible





With different starting point

Gradient descent Algorithm

$$\text{repeat until convergence}\{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \; \forall j \in \{0, 1\}\}$$

- Where
  - := is the assignment operator
  - α is the **learning rate** which basically defines how big the steps are during the descent
  - $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ is the **partial derivative** term
  - j = 0, 1 represents the **feature index number**

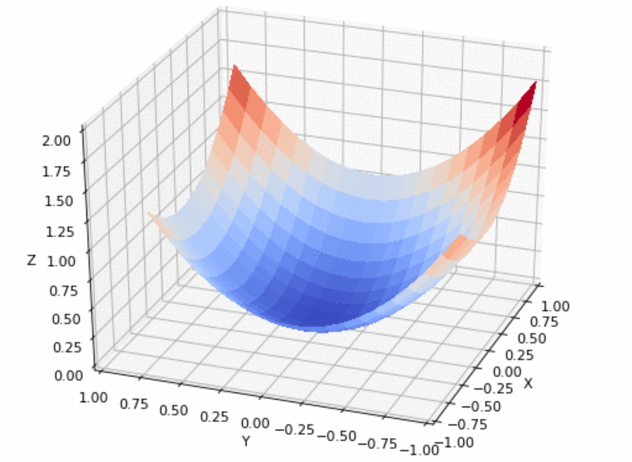
Also the parameters should be **updated simulatenously**, i.e. ,

$$temp_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

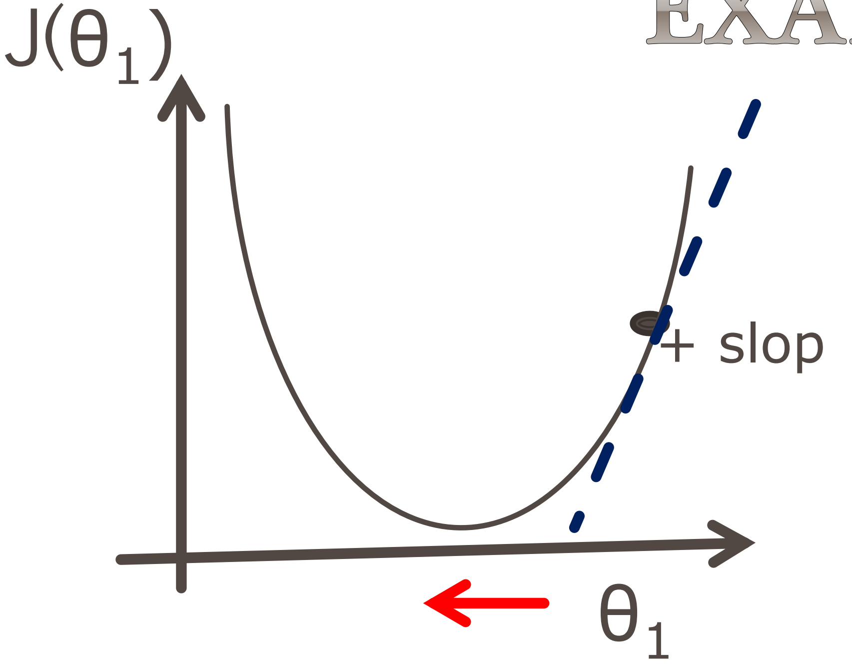
$$temp_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := temp_0$$

$$\theta_1 := temp_1$$

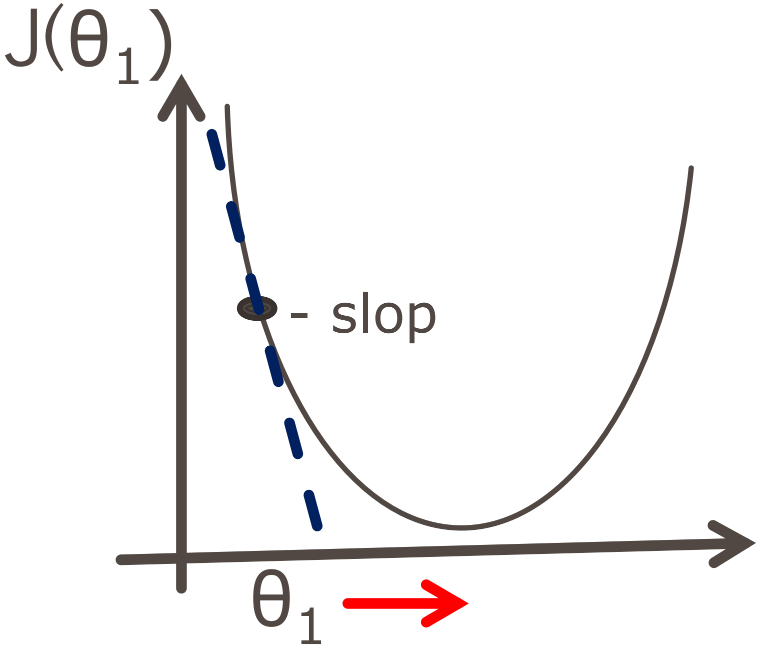


EXAMPLE



$$\boxed{\theta_1 = \theta_1 - \alpha \frac{d}{d\theta_1} j(\theta_1)}$$

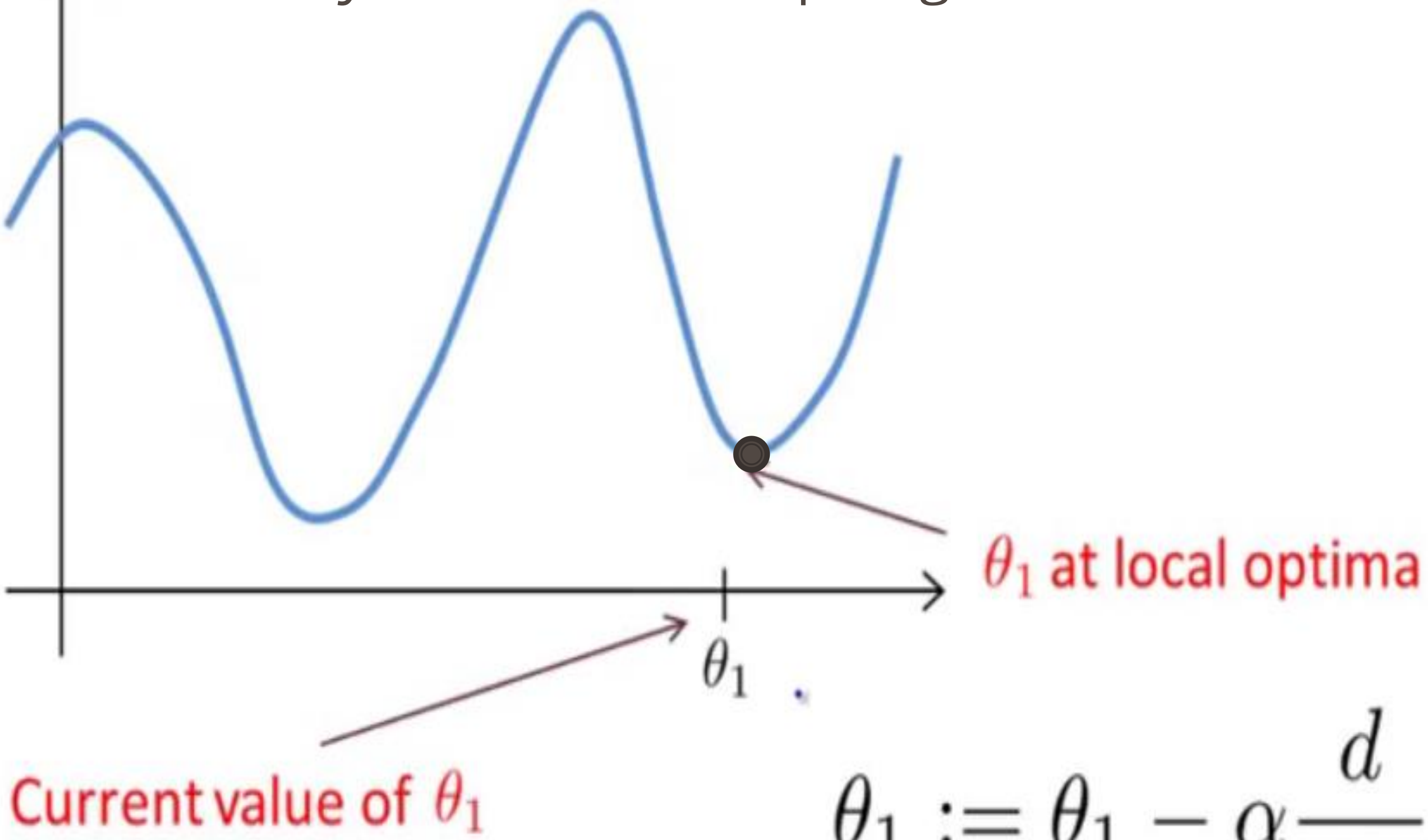
$\theta_1 = \theta_1 - \alpha(+ve)$



$\theta_1 = \theta_1 - \alpha(-ve)$

QUESTION

what do you think one step of gradient descent will do?



$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

GRADIENT DESCENT FOR A LINEAR REGRESSION

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$(\text{for } j = 1 \text{ and } j = 0)$$

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$\boxed{\frac{d}{d\theta_j} j(\theta_0, \theta_1) = \frac{d}{d\theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h\theta(x_i) - Y_i \right)^2}$$

$$\boxed{\frac{d}{d\theta_j} j(\theta_0, \theta_1) = \frac{d}{d\theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta 0 + \theta 1(x_i) - Y_i \right)^2}$$

$$j = 0 : \frac{d}{d\theta_0} j(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - Y_i \right)$$

$$j = 1 : \frac{d}{d\theta_1} j(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - Y_i \right) \bullet x_i$$

G.D. FOR LINEAR REGRESSION

Gradient descent algorithm

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

}

"Batch" Gradient Descent

"Batch": Each step of gradient descent uses all the training examples.

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

Linear Regression

Using

TensorFlow

1-D Data Example

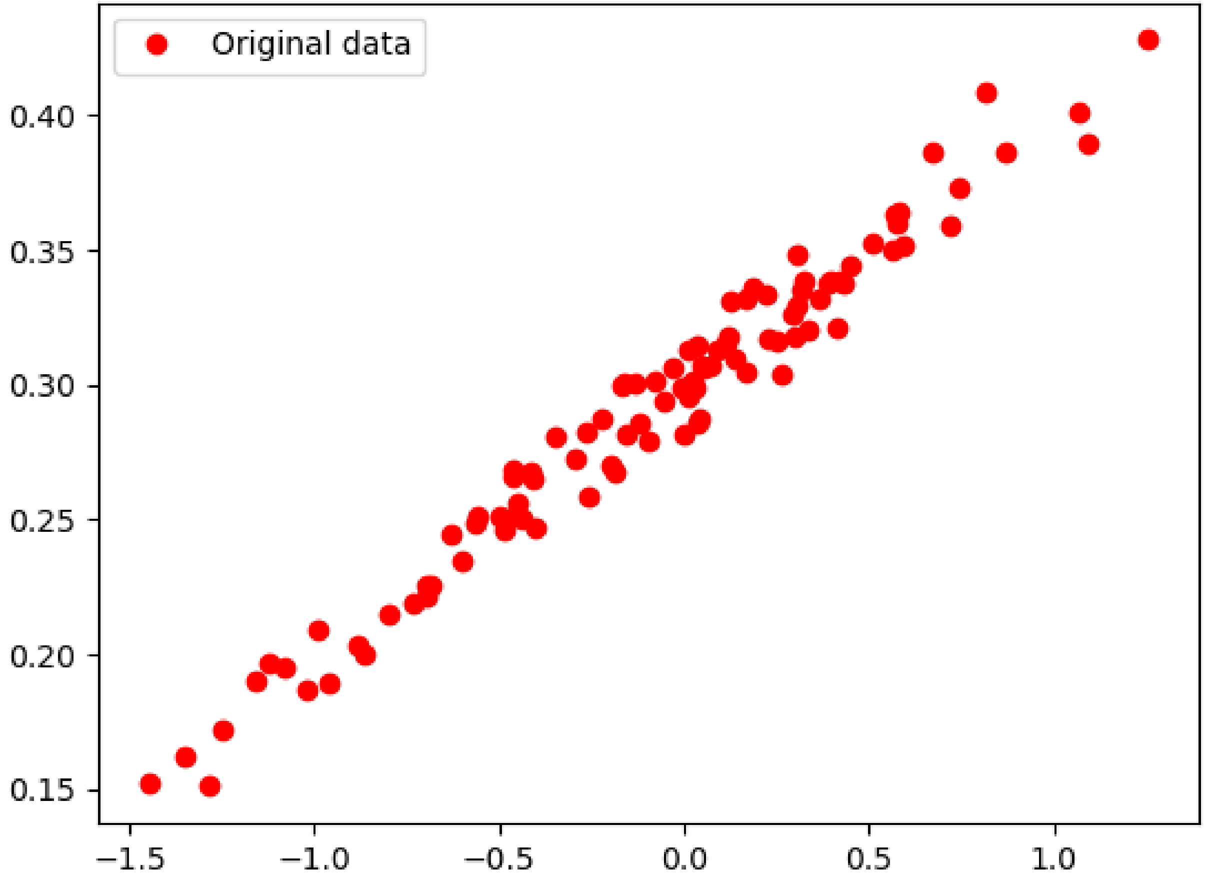
Data Preparation

```
import numpy as np

    num_of_points = 100         #Generate 100 Data Points
    points = []
    for i in range(num_of_points):
        x1= np.random.normal(0.0, 0.55)
        y1= x1 * 0.1 + 0.3 + np.random.normal(0.0, 0.01)
        points.append([x1, y1])
    x_data = [v[0] for v in points]
    y_data = [v[1] for v in points]
```

Draw Data

```
import matplotlib.pyplot as plt

plt.plot(x_data, y_data, 'ro', label='Original data')
plt.legend()
plt.show()
```

Original Data



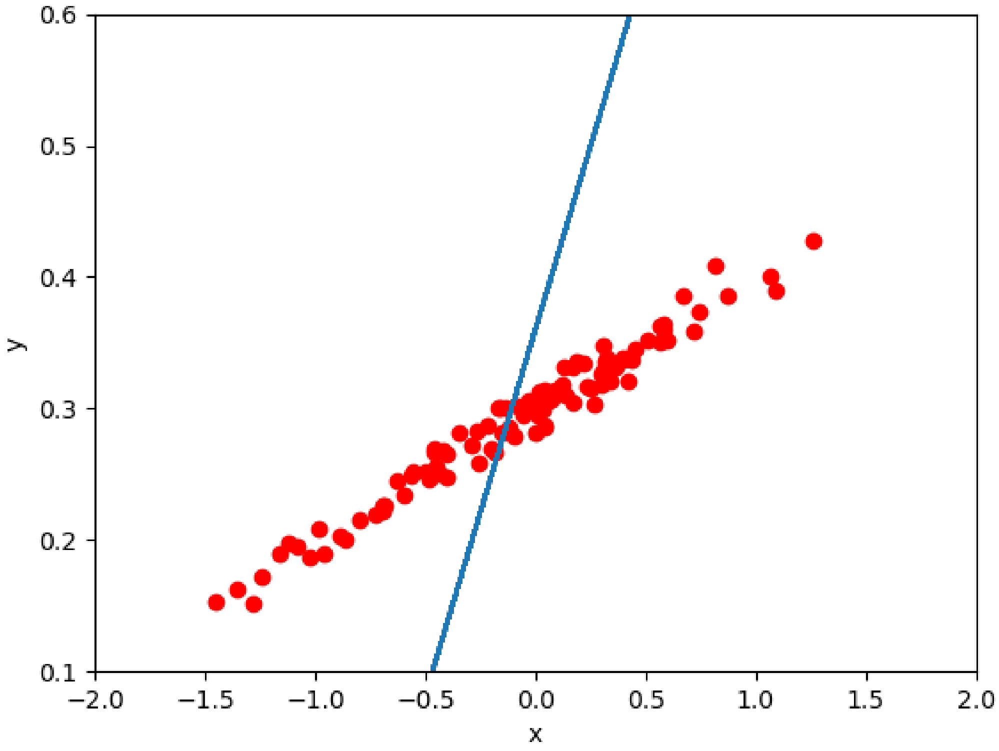
Variables and Nodes Preparation

```
import tensorflow as tf

#initialize weights "W and bias "b"
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))

y = W * x_data + b

#Define Loss function as Mean of Squared Error
loss = tf.reduce_mean(tf.square(y - y_data))

#Create Optimizer class to minimize Losses
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

#initialize TensorFlow Variables (always)
init = tf.global_variables_initializer()
```

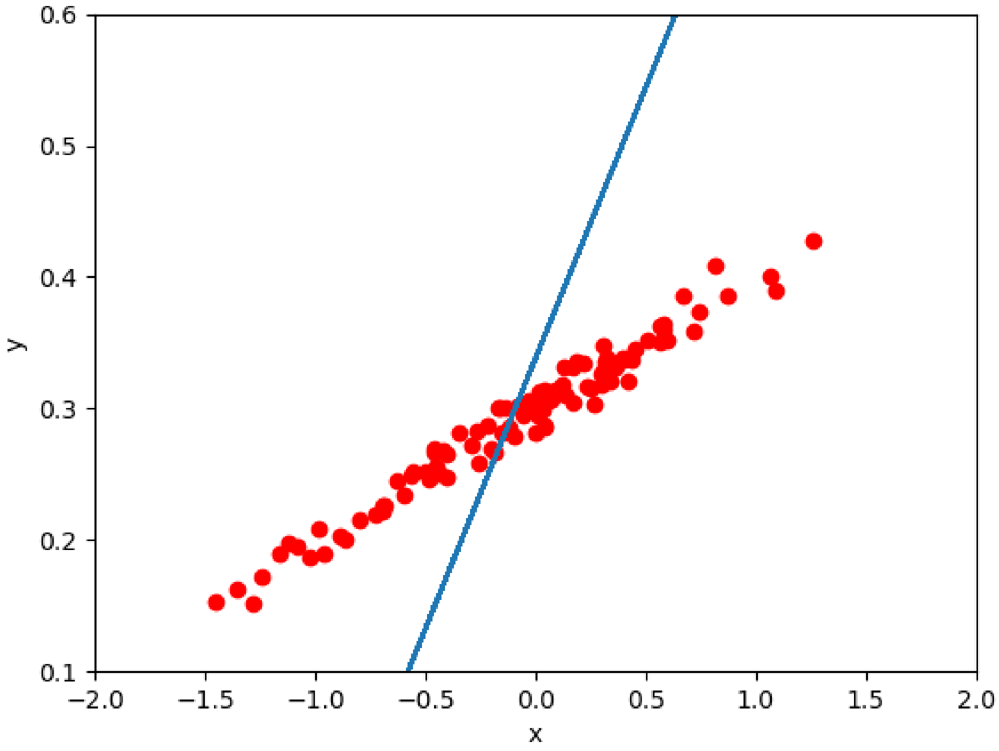
Execute TensorFlow Graph

```
#Start TensorFlow Session and carryout Variable initialization
sess = tf.Session()
sess.run(init)

#Carryout 16 Iterations
for step in range(16):
    sess.run(train)

    #Draw Original Data
    plt.plot(x_data, y_data, 'ro')

    #Draw Predicted data (using calculated weight and bias after training
    plt.plot(x_data, sess.run(W) * x_data + sess.run(b))
    plt.xlabel('x')
    plt.xlim(-2, 2)
    plt.ylim(0.1, 0.6)
    plt.ylabel('y')
    plt.legend()
    plt.show()

   # print updated weights, bias, and Loss value after current training iteration
    print(step, sess.run(W), sess.run(b),sess.run(loss))
```

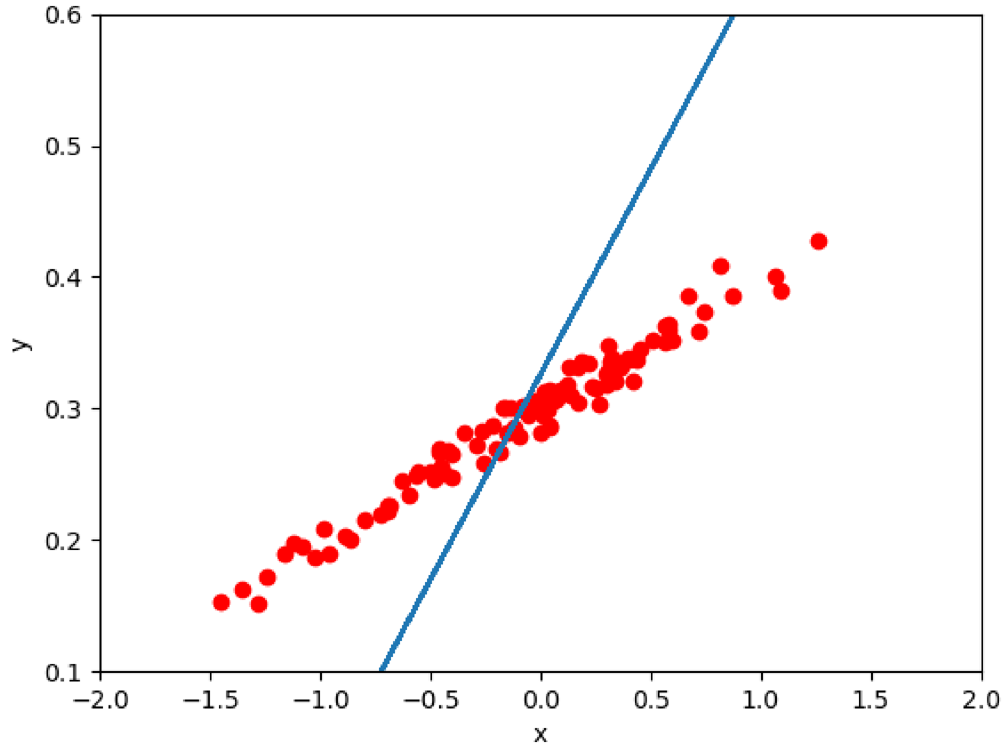
Iteration 1



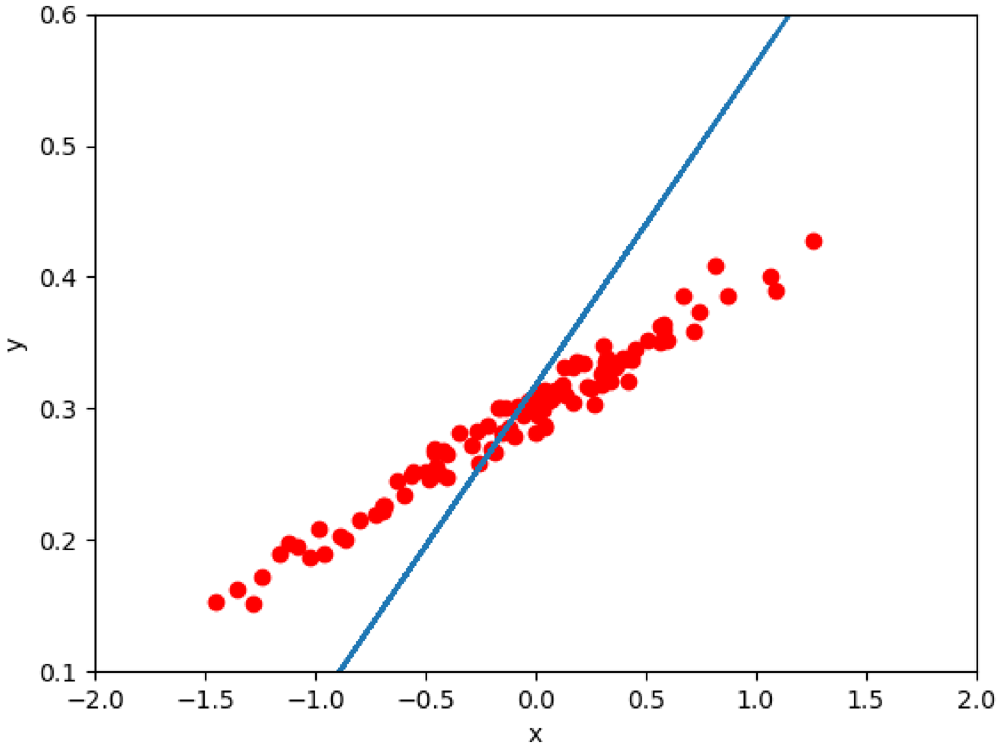
Iteration 2



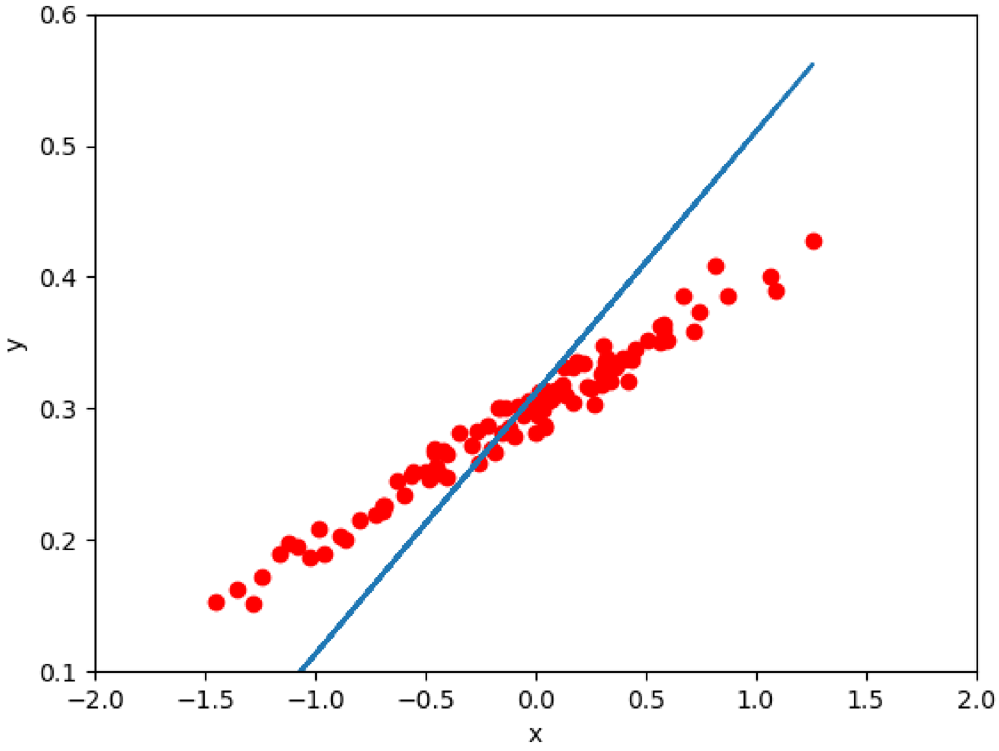
Iteration 3



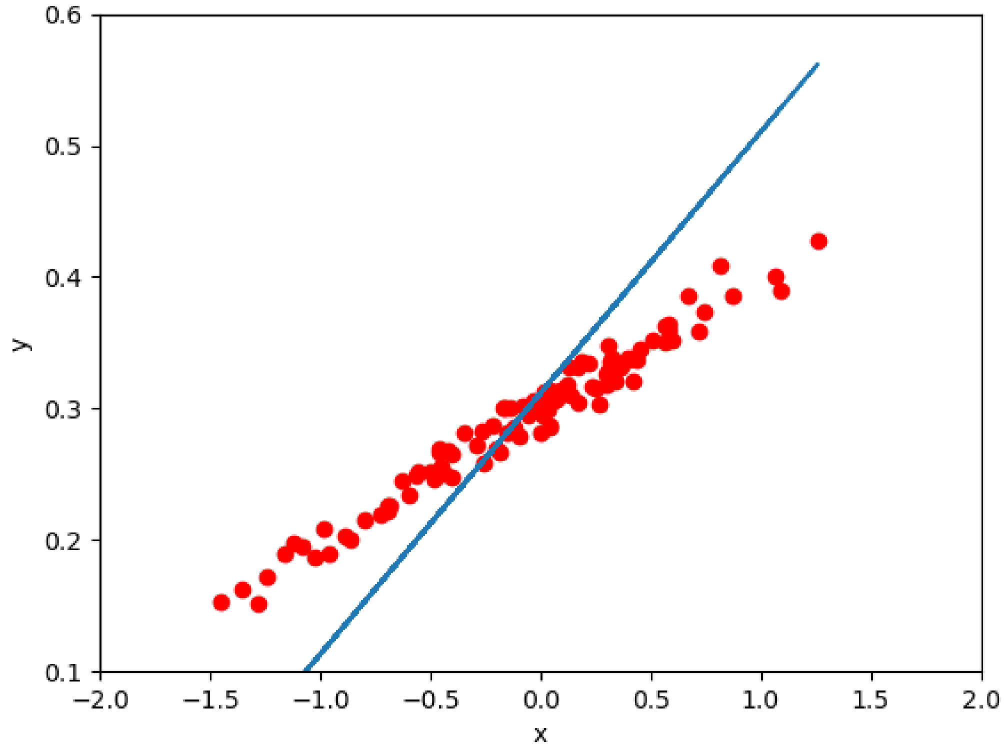
Iteration 4



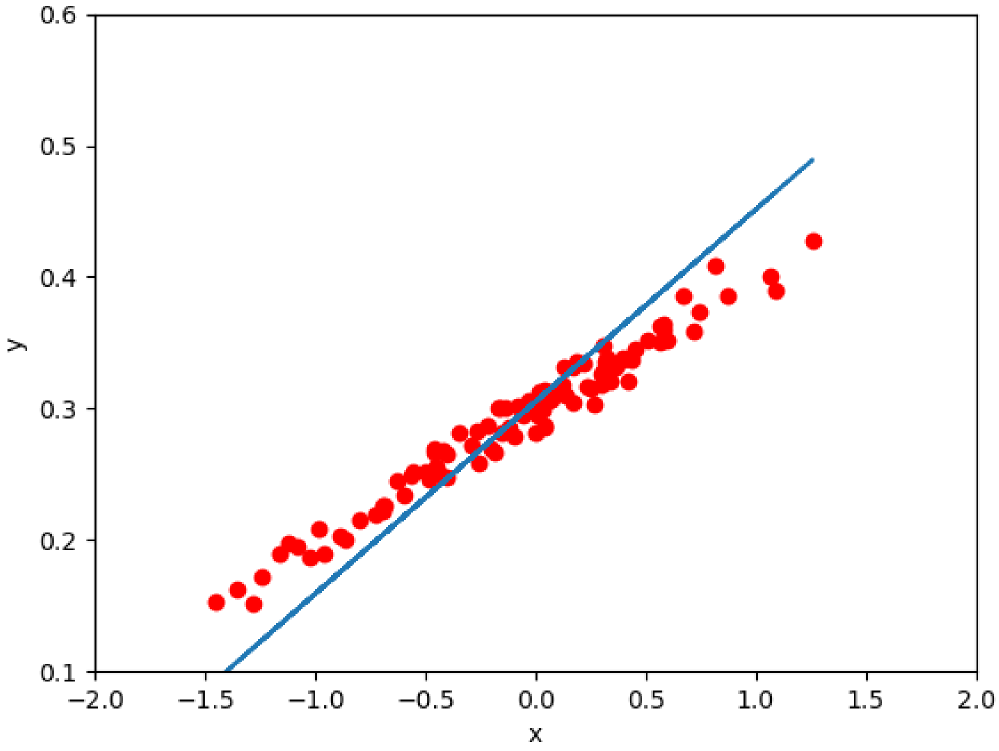
Iteration 5



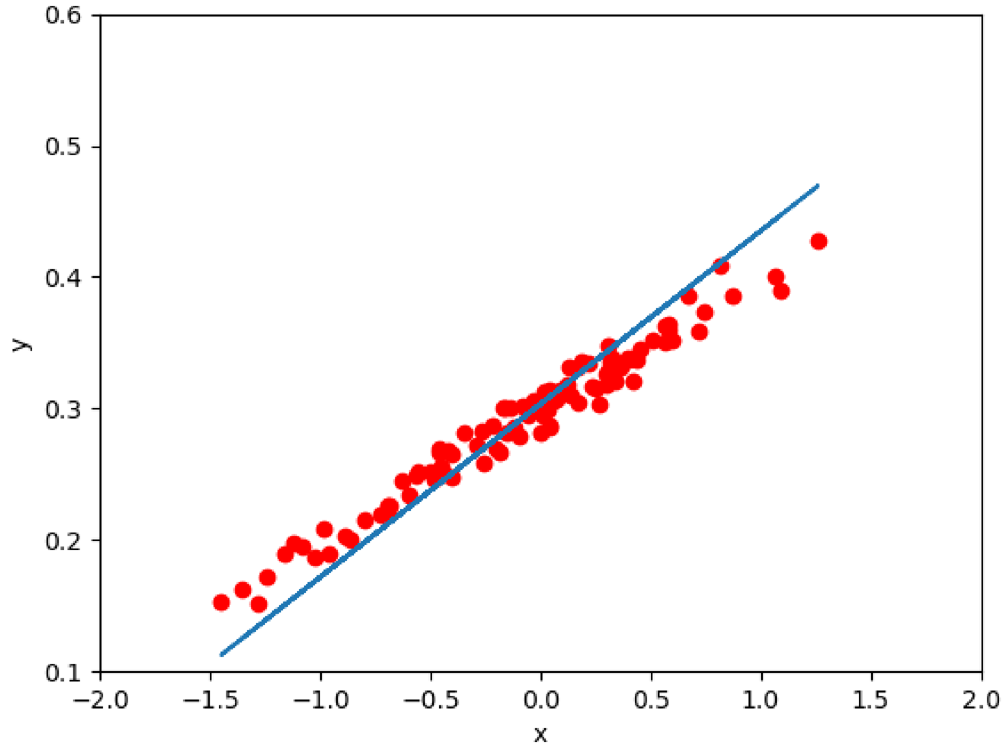
Iteration 6



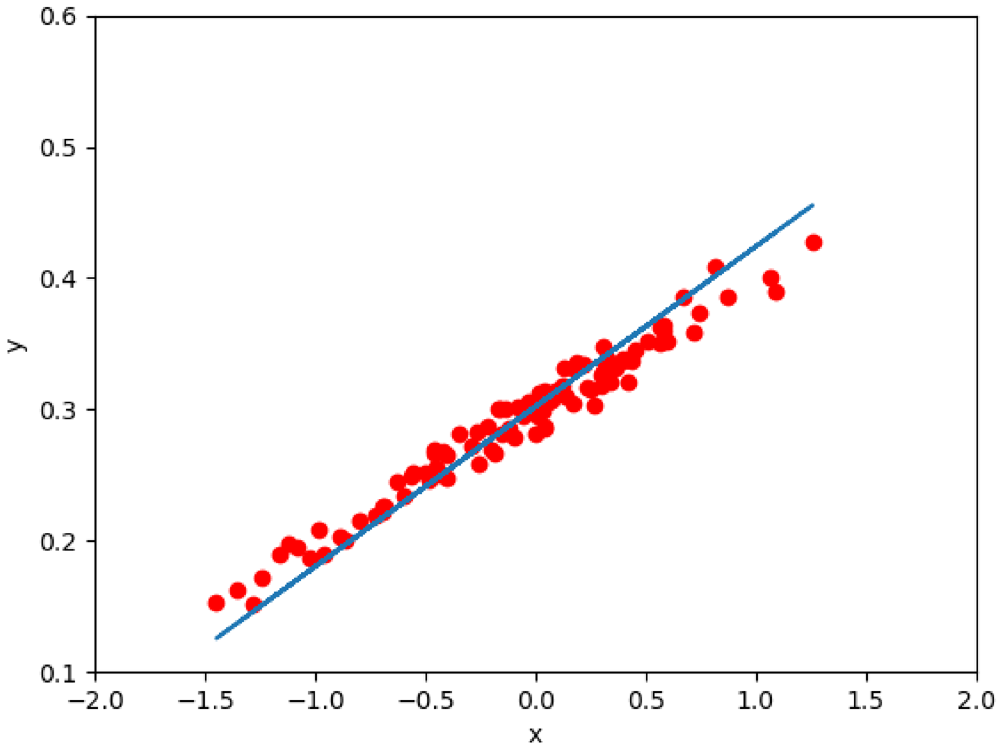
Iteration 7



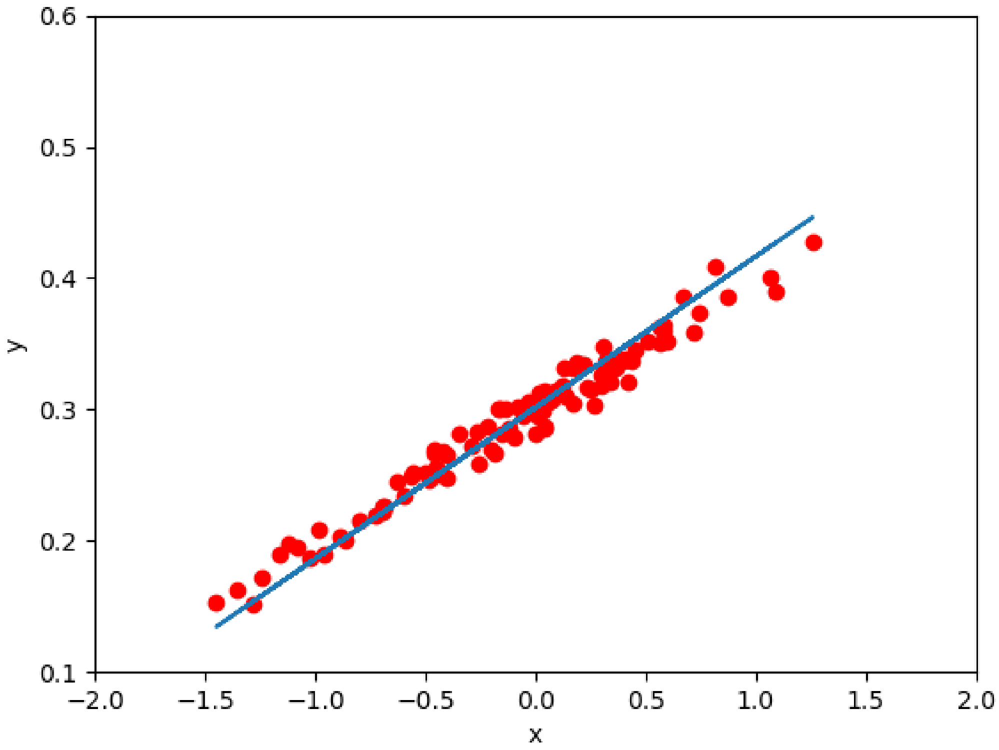
Iteration 8



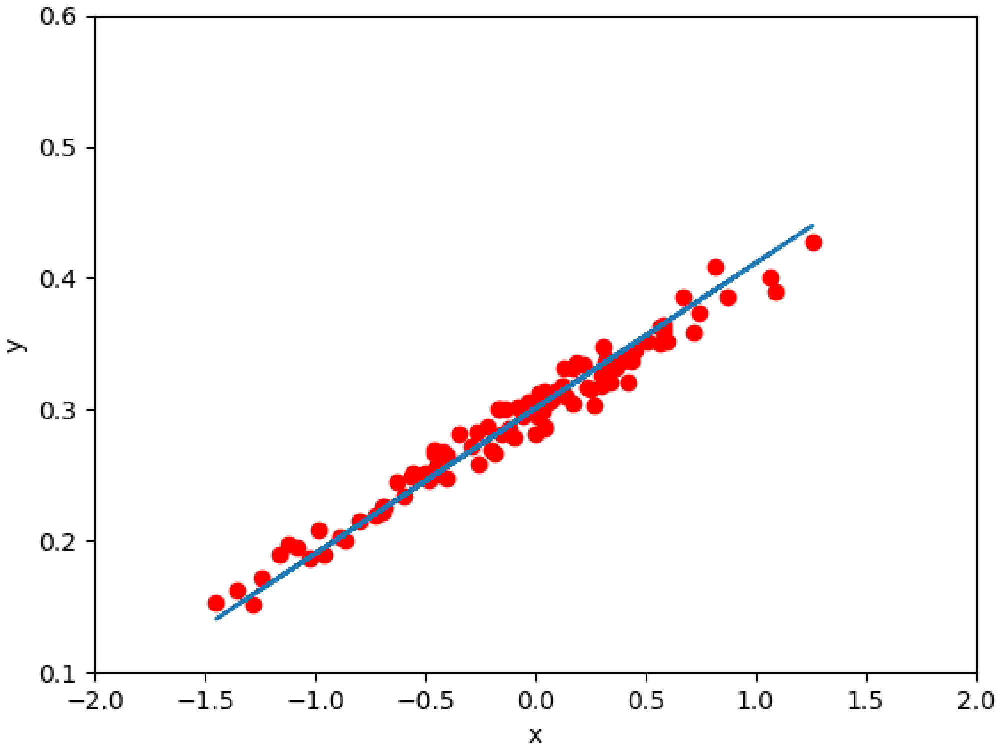
Iteration 9



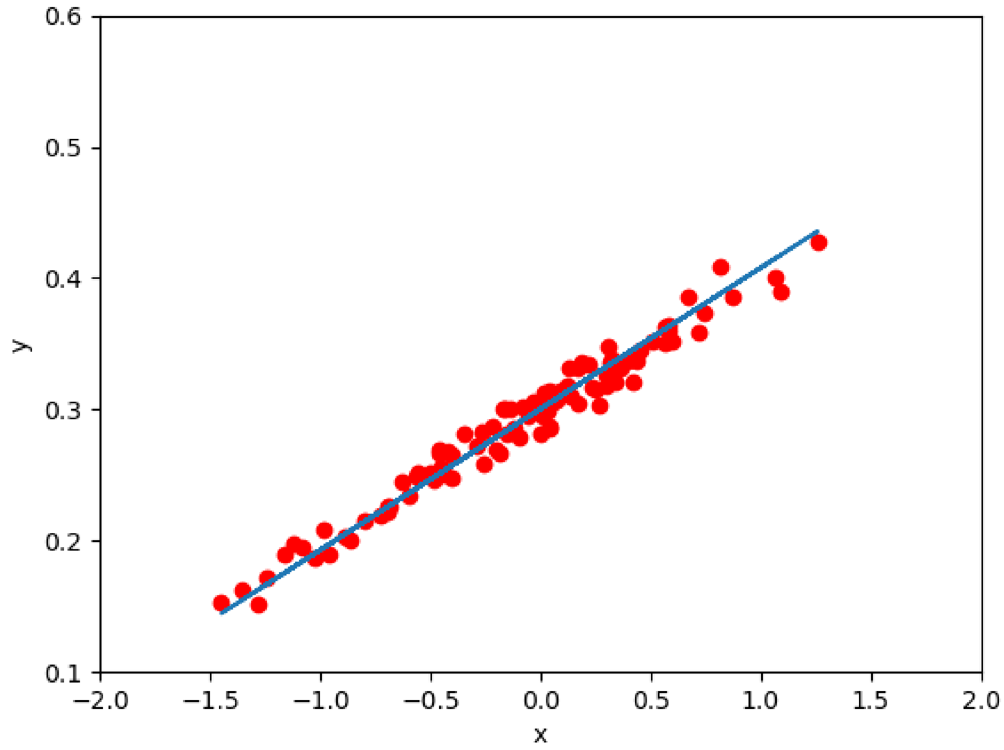
Iteration 10



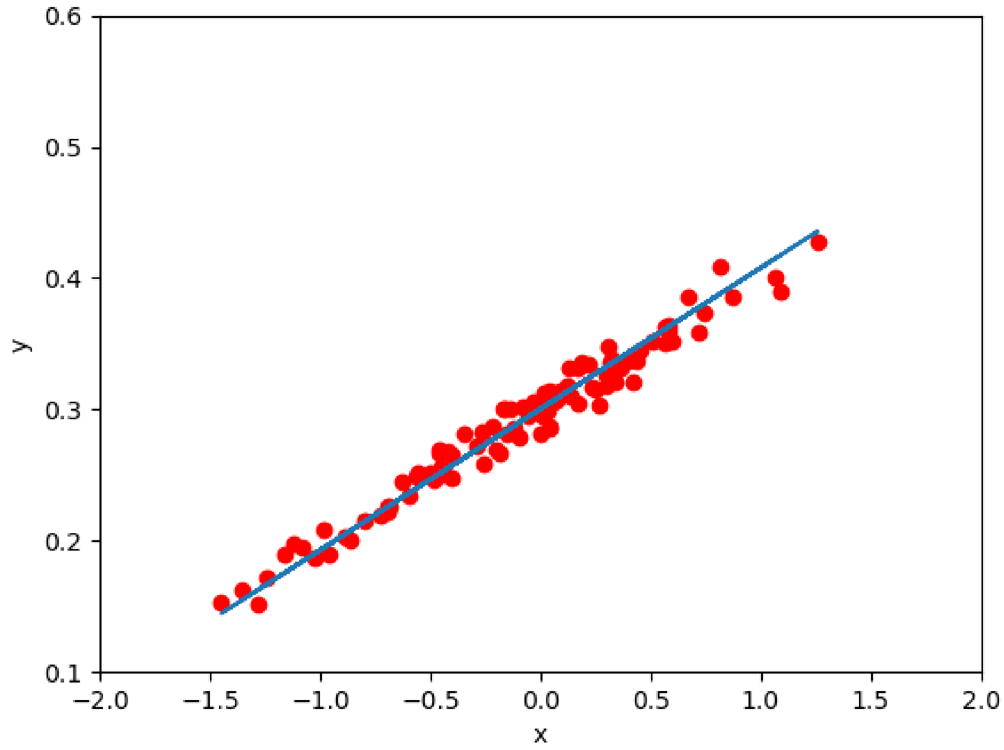
Iteration 11



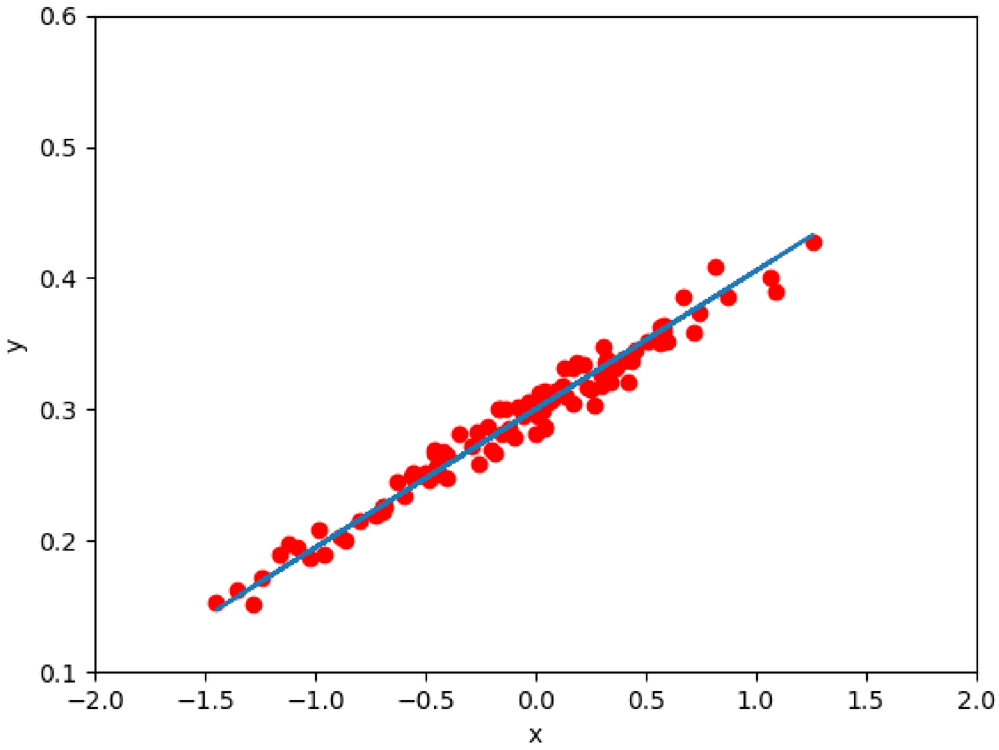
Iteration 12



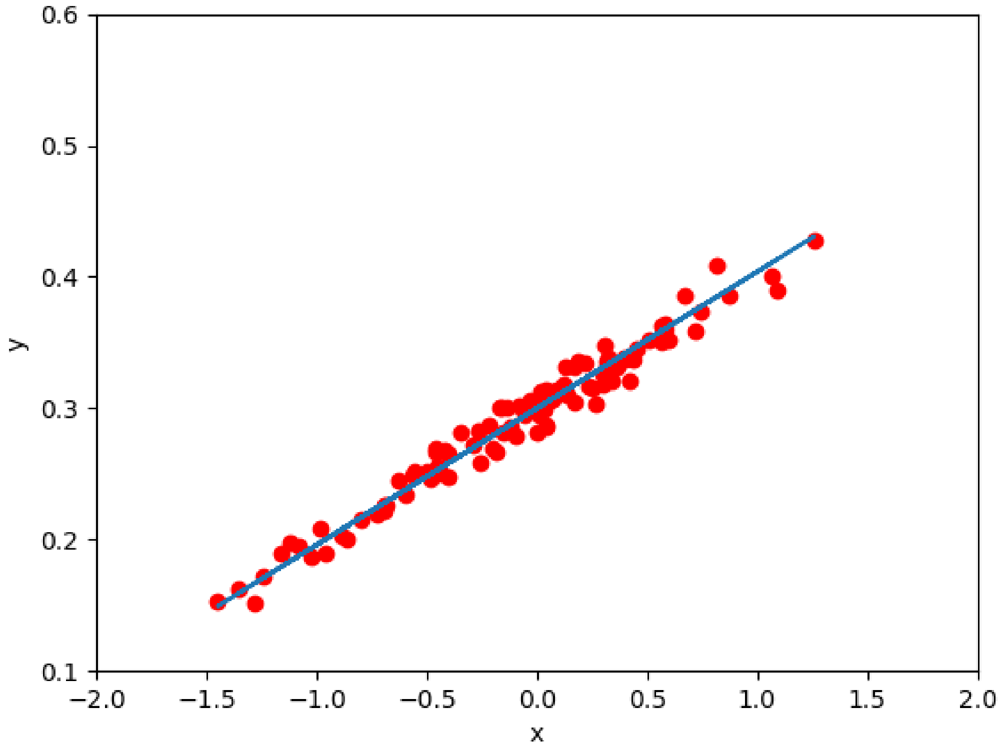
Iteration 13



Iteration 14



Iteration 15



Iteration 16