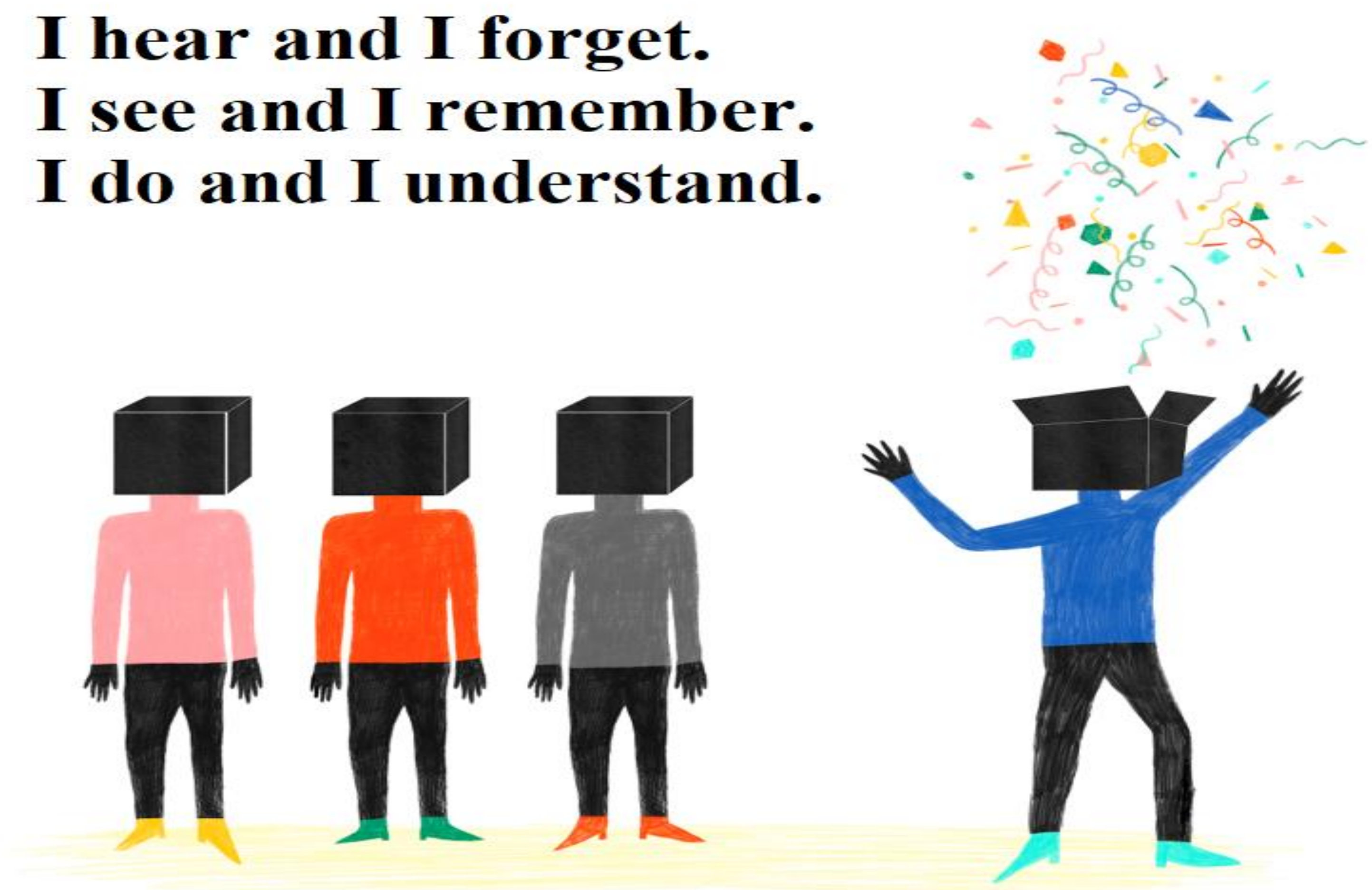


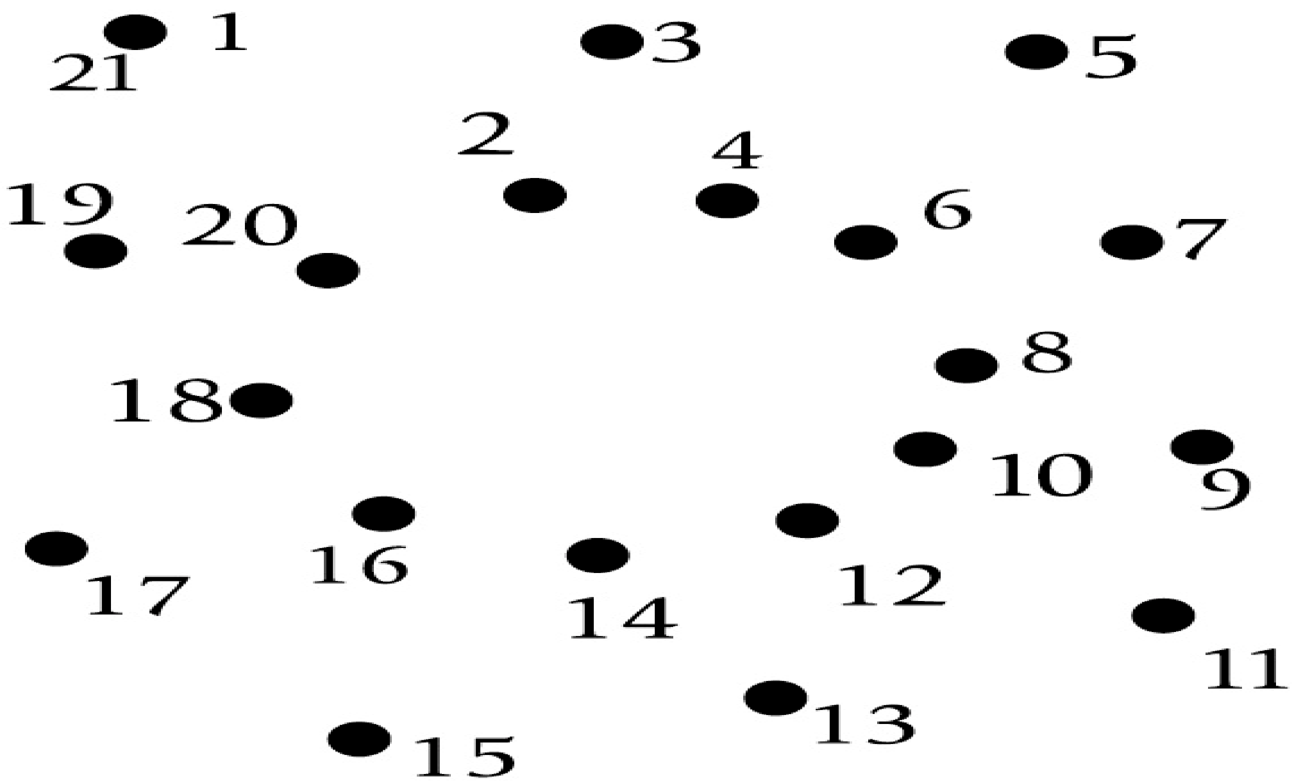
Chandan Verma

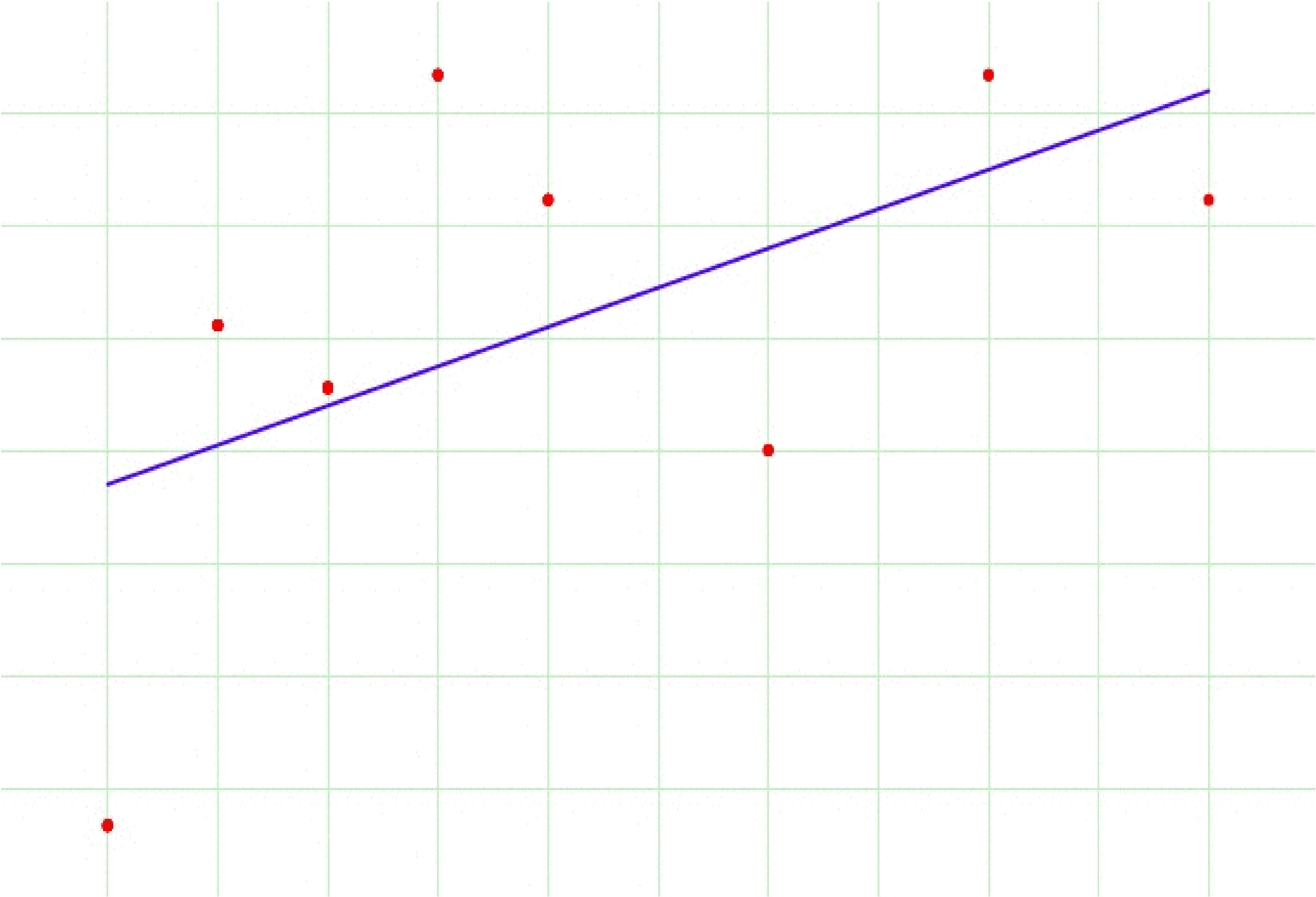
Corporate Trainer(Machine Learning,AI,Cloud Computing,IOT)

www.facebook.com/verma.chandan.070

Curve fitting

Curve fitting, essentially, is similar to the game of connecting the dots where you try to complete a picture





0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267

9	2.4	4	9.2	?

Regression is the process of predicting a Continuous Value

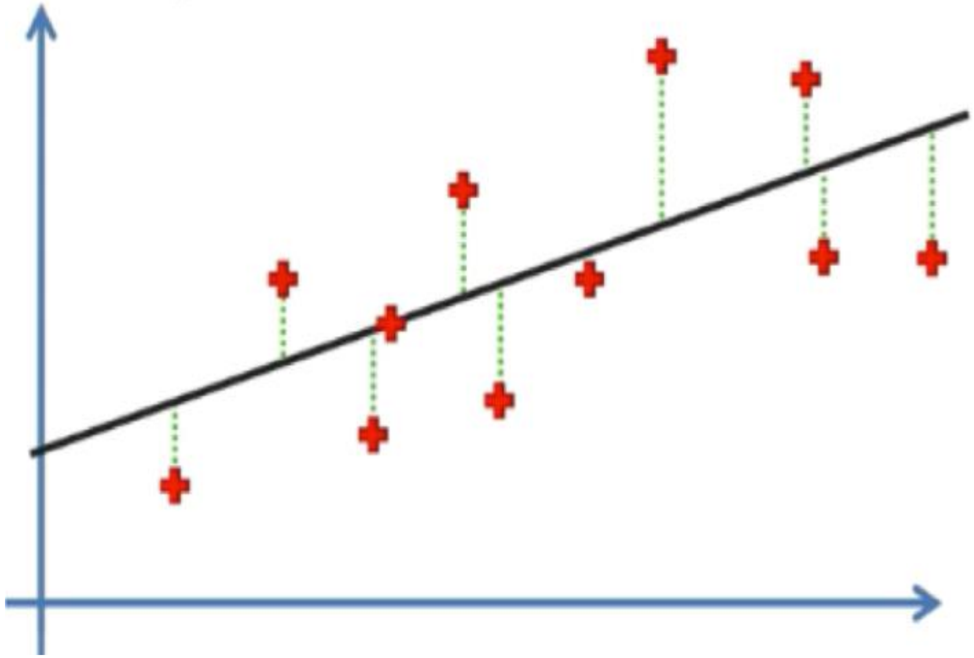
Linear Regression

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

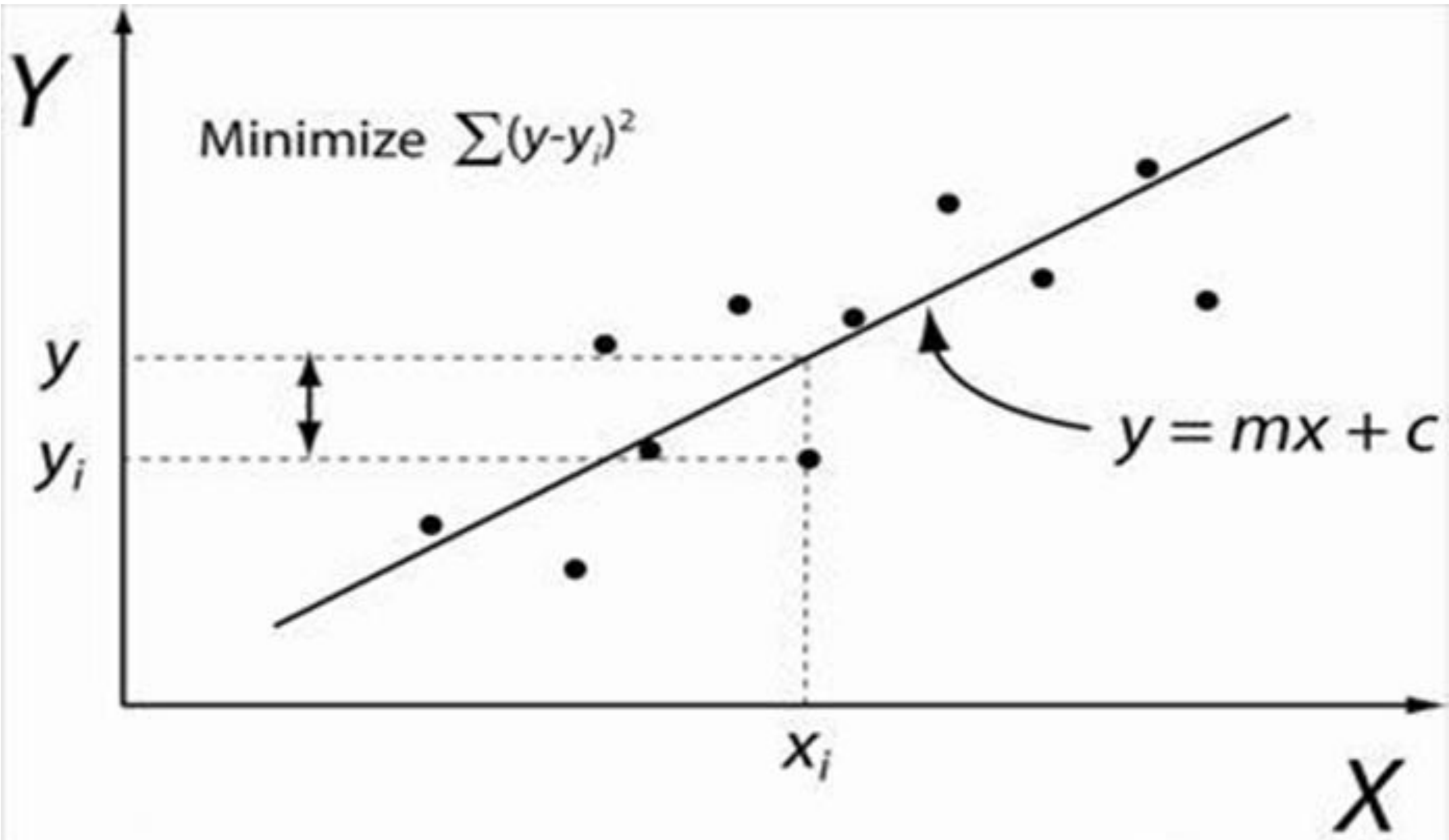
Regression is a statistical way to establish a relationship between a dependent variable and a set of independent variable(s).

Regression

Year	Sales (Million Euro)	Advertising (Million Euro)
1	651	23
2	762	26
3	856	30
4	1,063	34
5	1,190	43
6	1,298	48
7	1,421	52
8	1,440	57
9	1,518	58



Linear Regression

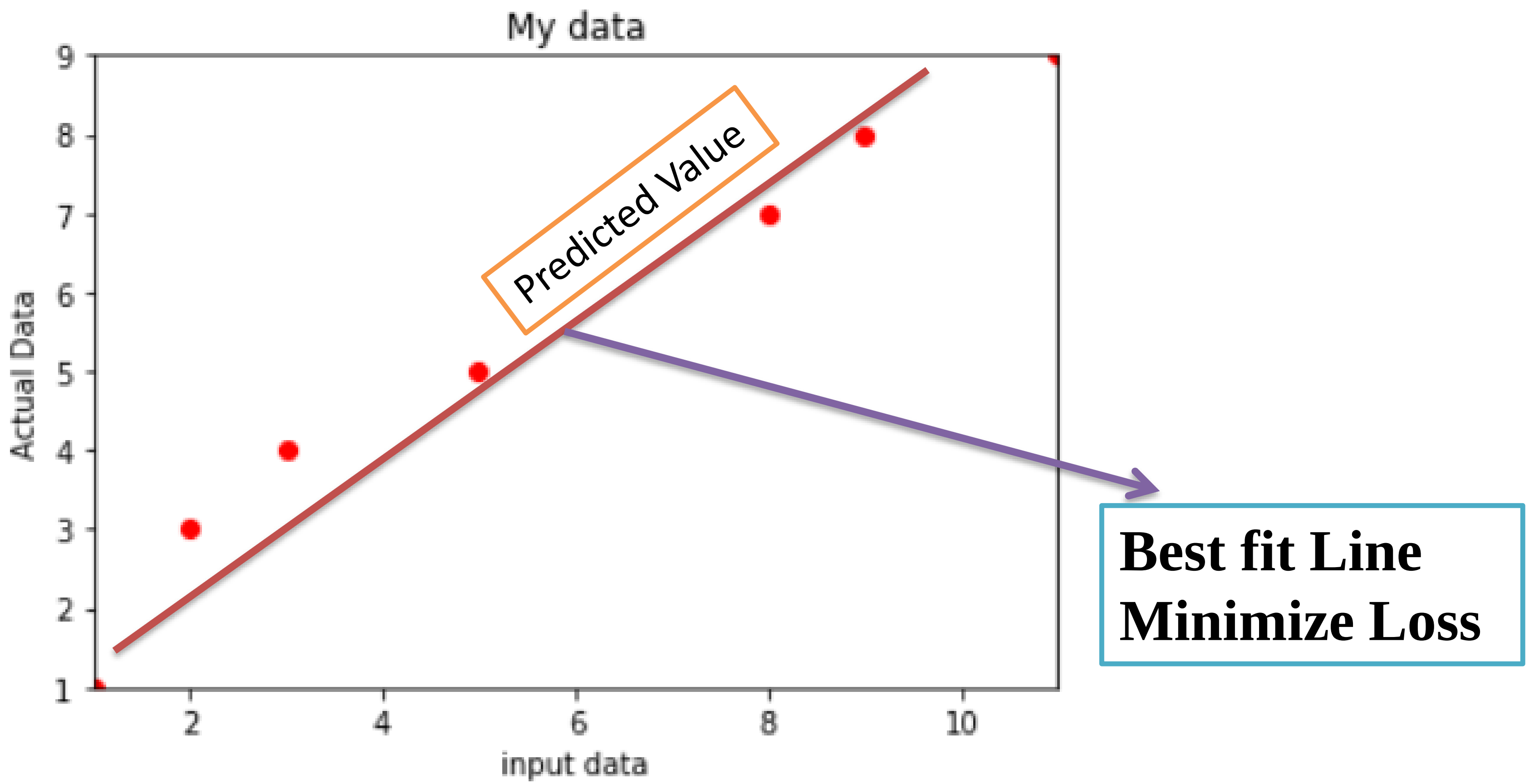


Linear Regression

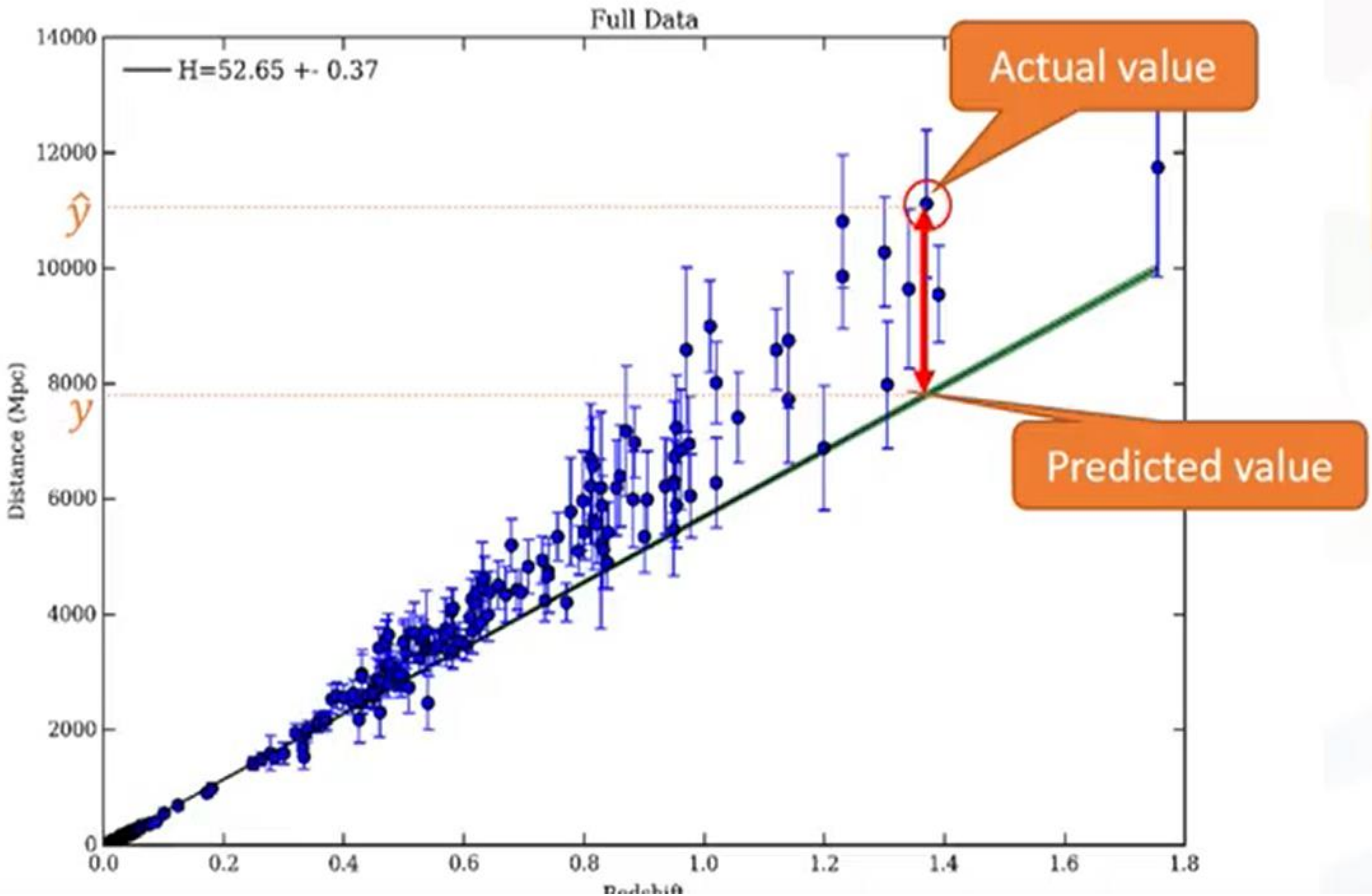
Linear regression is a statistical approach for modeling relationship between a **dependent(response)**variable with a given set of **independent(features)** variables.

special case of fitting a straight line to some given data

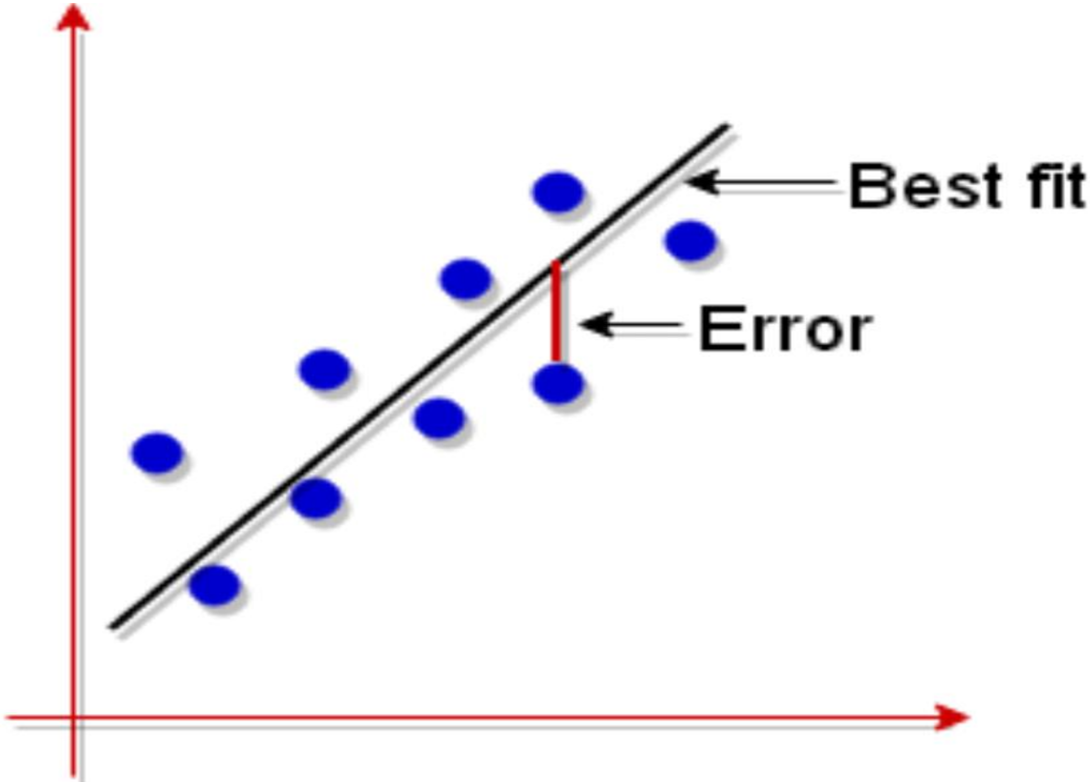
In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X.



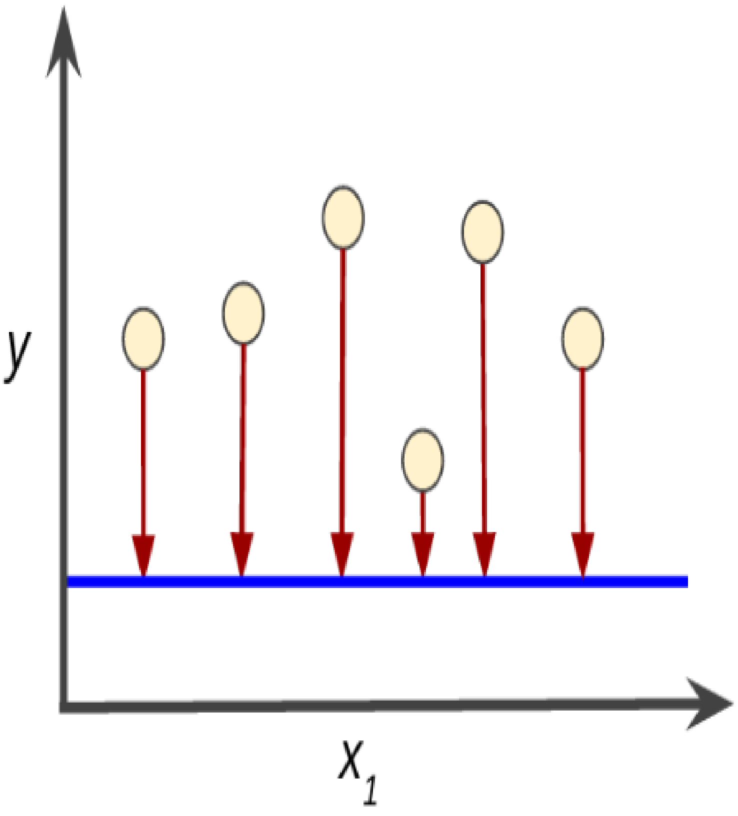
Loss=(Actual –Predicted)

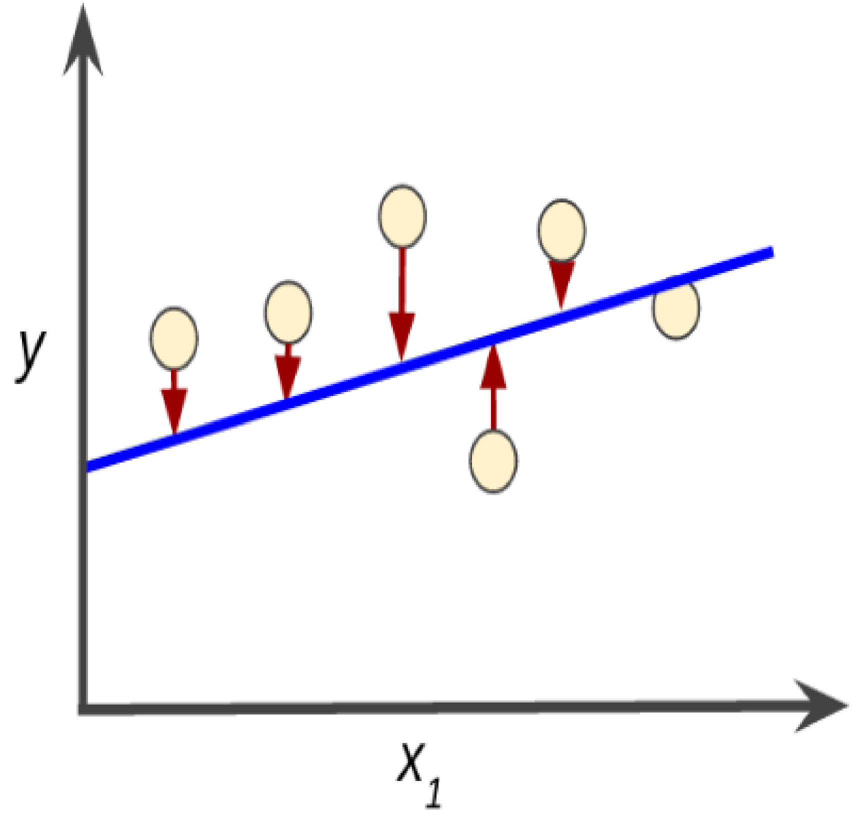


Full Data
H=52.65 +- 0.37
Actual value
Predicted value
$\hat{y}$
y
Distance (Mpc)
14000
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8

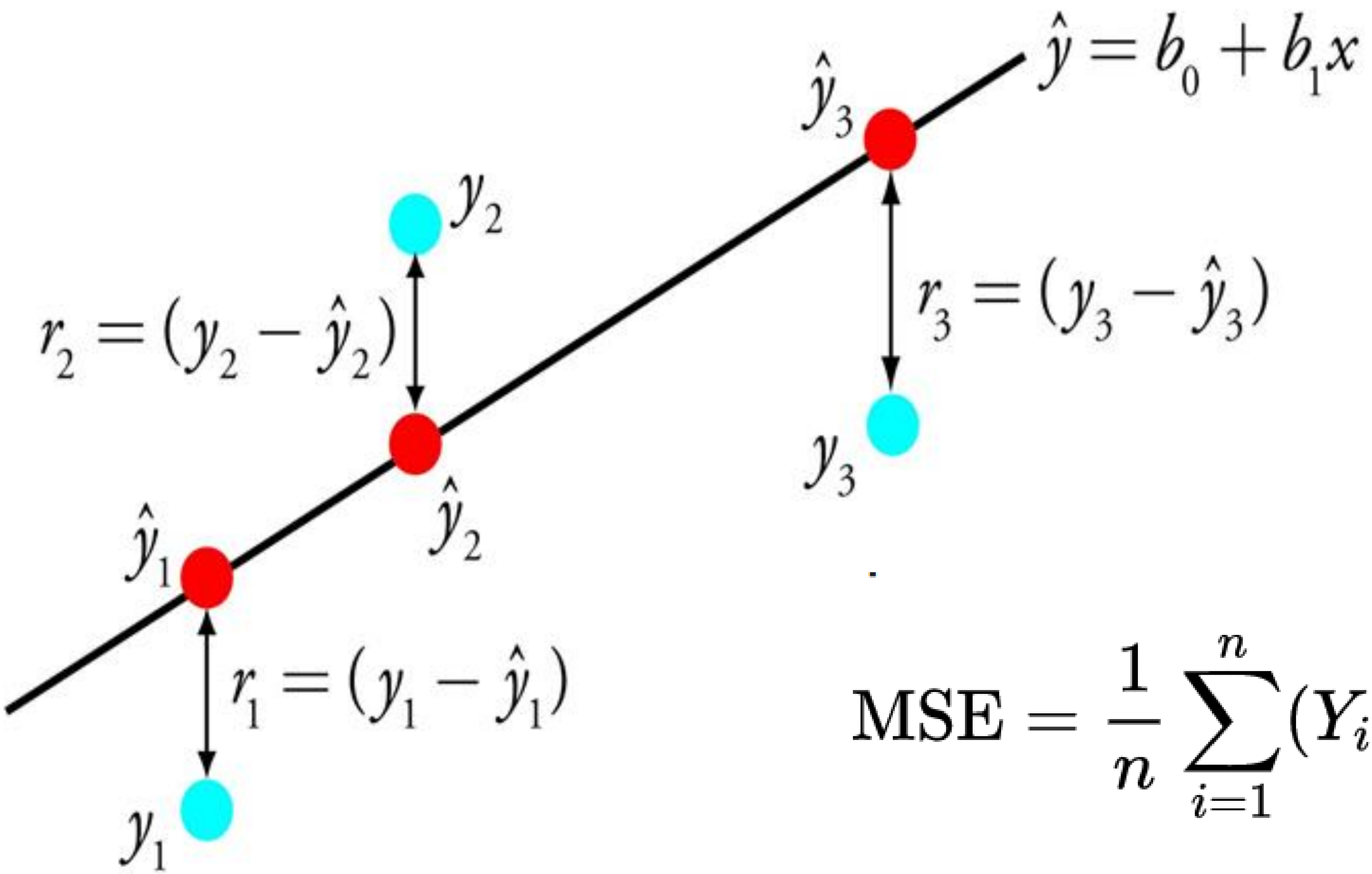


It is a method to predict the target variable by finding a best fit line between the independent and dependent variable
The best fit is the line with minimum error from all the Point





The technique we will use to find the *best fitting* line will be called the *method of least squares*.



$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Best fit line

Divide by the total number of data points

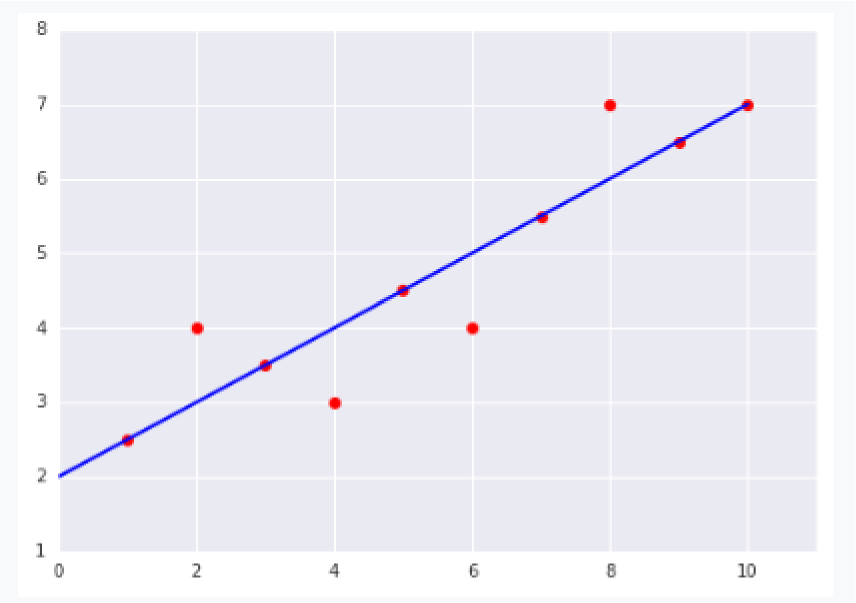
Predicted output value

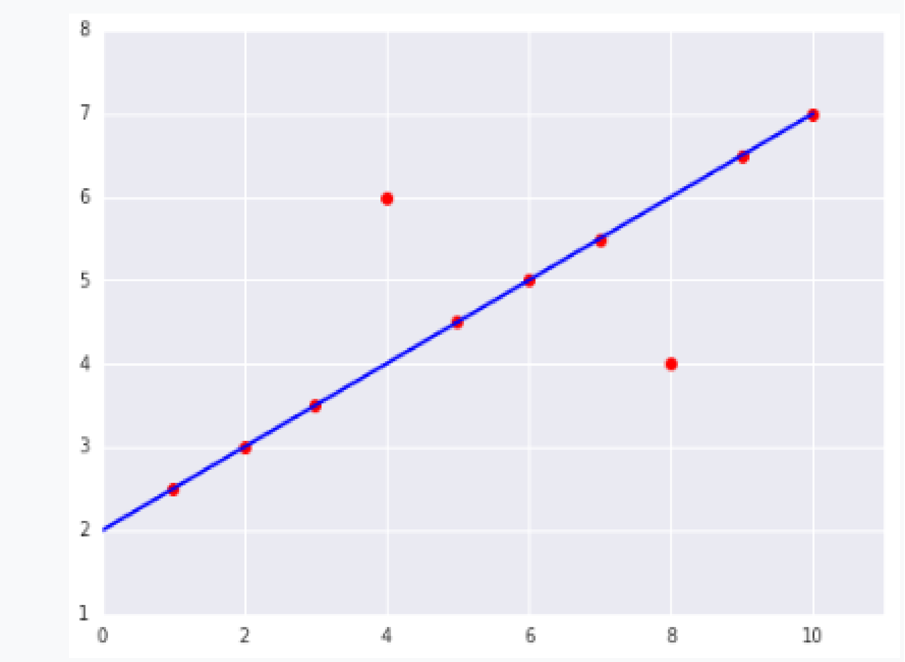
Actual output value

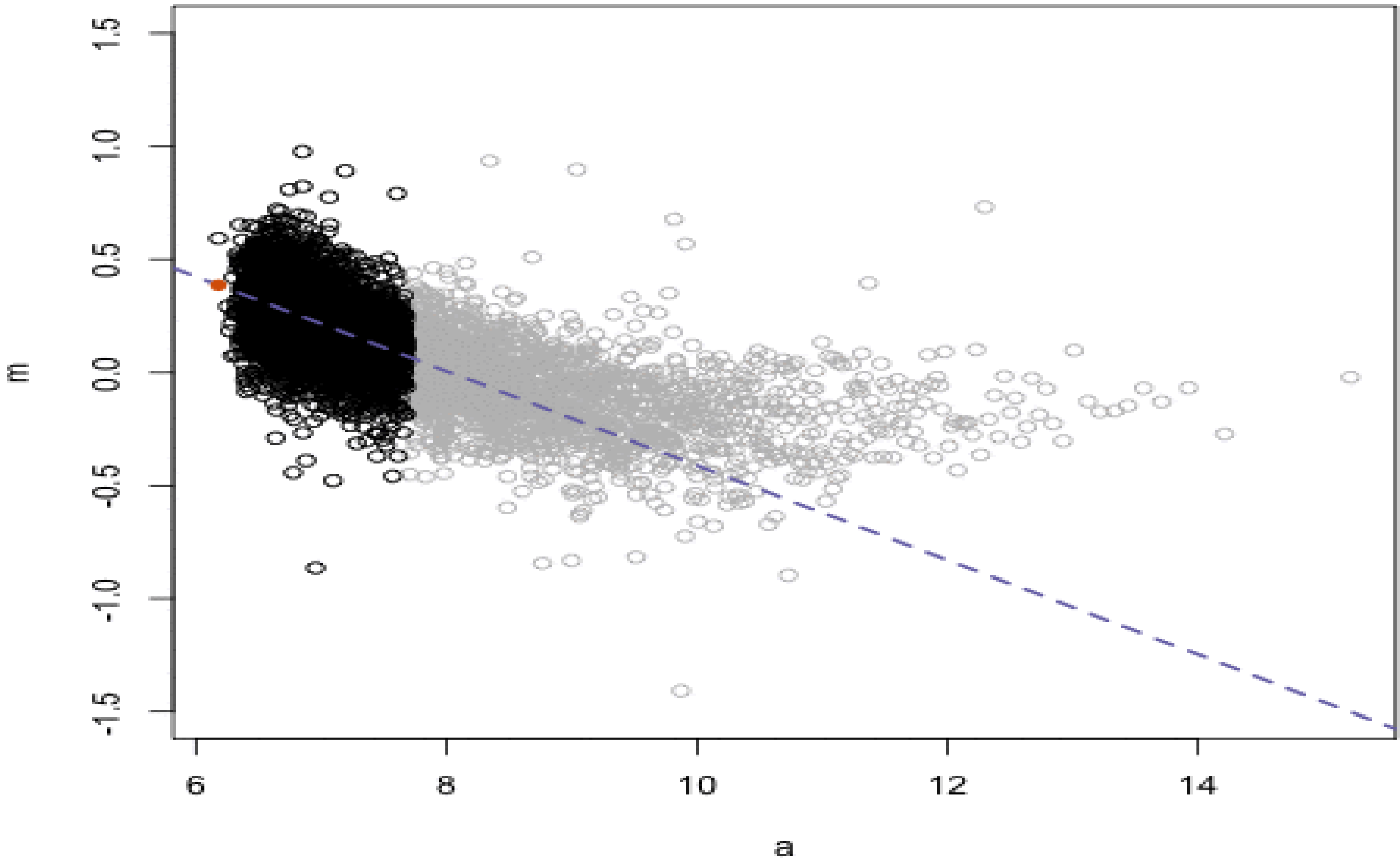
$$MAE = \boxed{\frac{1}{n}} \underset{\text{Sum of}}{\sum} \underbrace{\left| \boxed{y} - \boxed{\hat{y}} \right|}_{\text{The absolute value of the residual}}$$

There are simple linear regression calculators that use a “least squares” method to discover the best-fit line for a set of paired data. You then estimate the value of X (dependent variable) from Y (independent variable

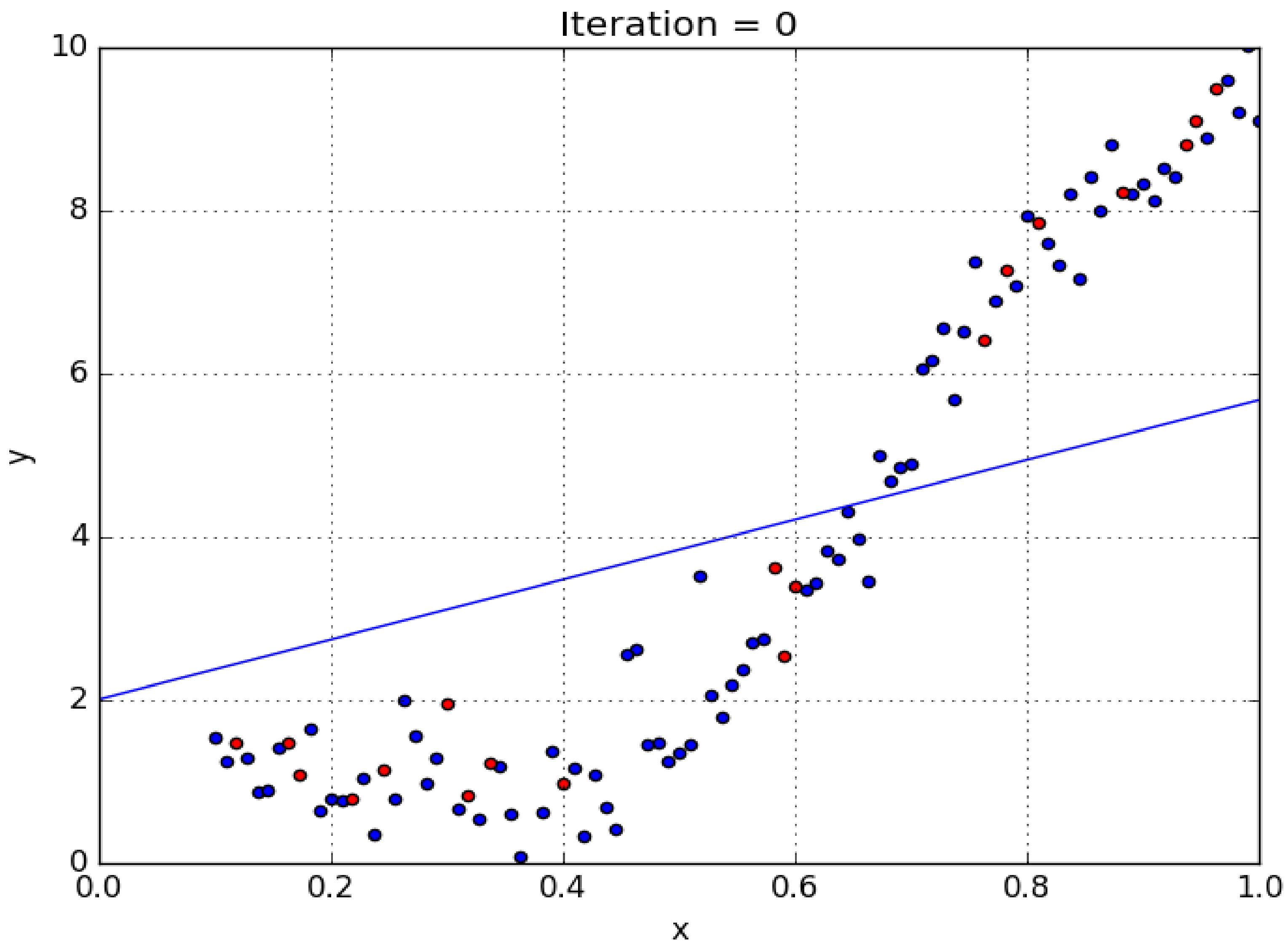
Best fit line



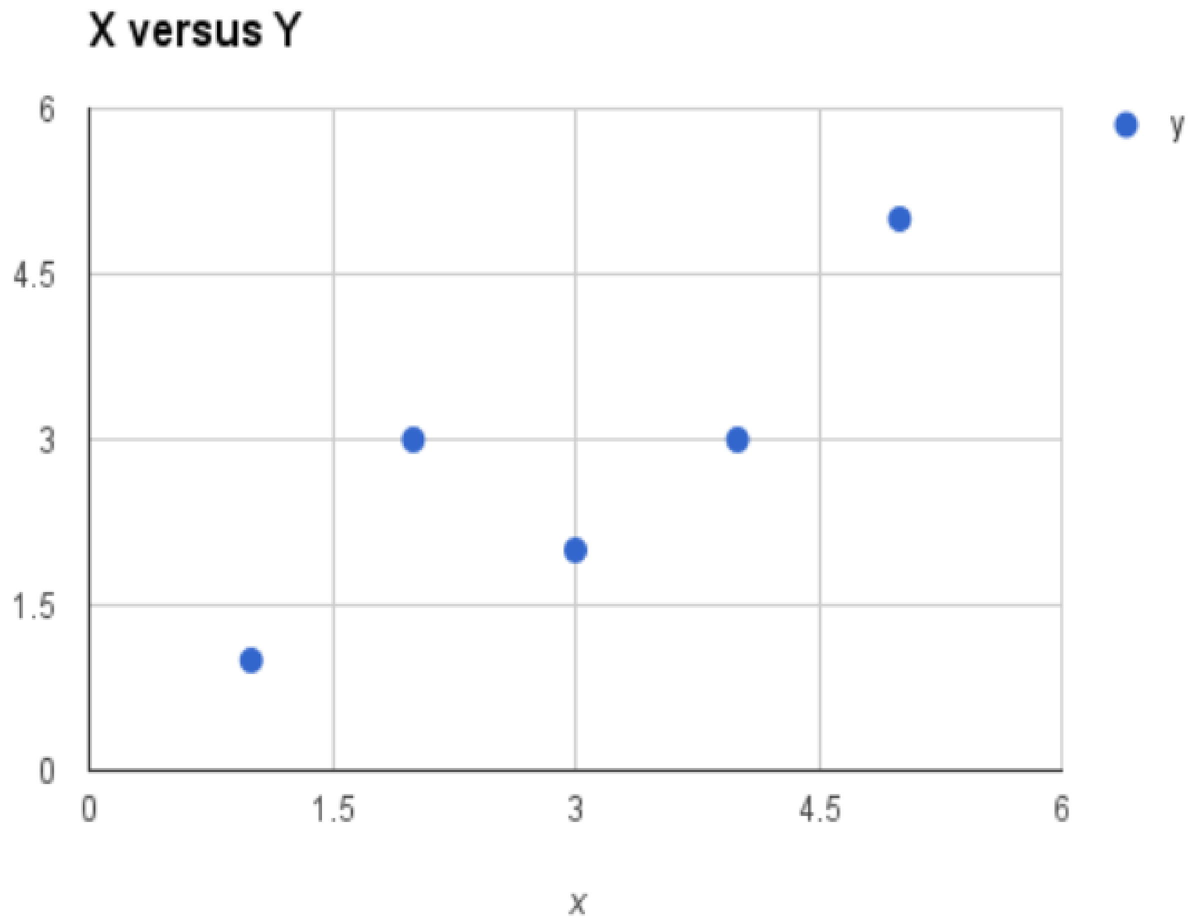




1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
m
6
8
10
12
14
a


Iteration = 0
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0
x
y

x	y
1	1
2	3
4	3
3	2
5	5



$$y = B0 + B1 * x$$

Simple Linear Regression

1. Mean Function.
2. Variance Function.
3. Covariance Function.
4. Functions to calculate the and values

$$w_1 = \frac{covariance(x,y)}{variance(x)}$$

$$w_0 = mean(y) - (w_1 * mean(x))$$

Simple Linear Regression

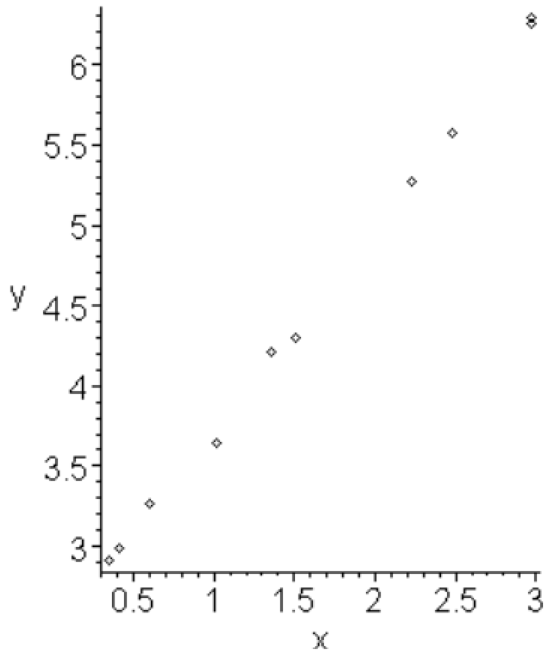
Having one independent variable to predict the dependent variable.

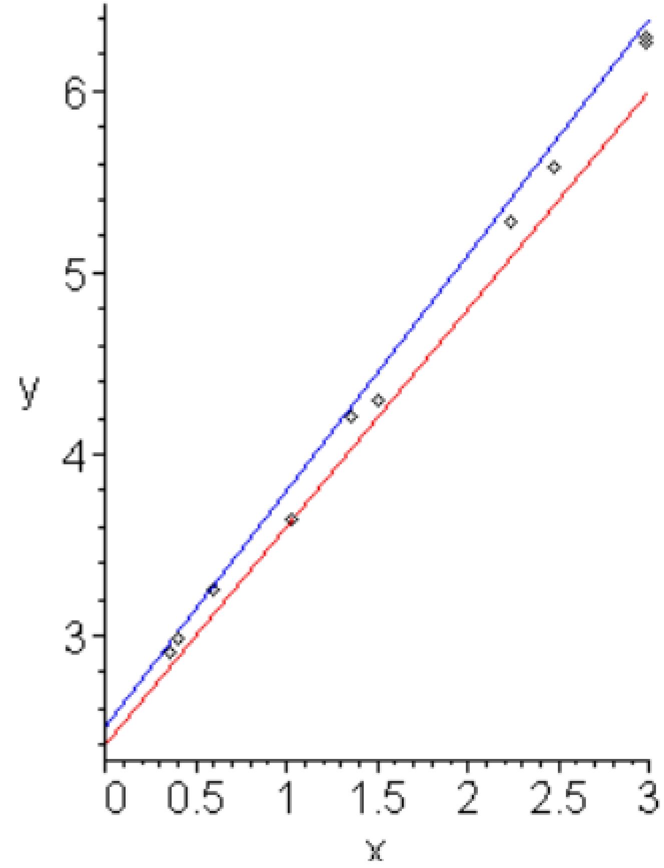
$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - mean(x))^2}{n-1}$$

$$cov_{x,y} = \frac{\sum_{i=1}^{N} (x_i - mean(x))(y_i - mean(y))}{N-1}$$

$$\text{mean}(x) = \frac{(x_1) + (x_2) + (x_3) \ldots + (x_n)}{n}$$

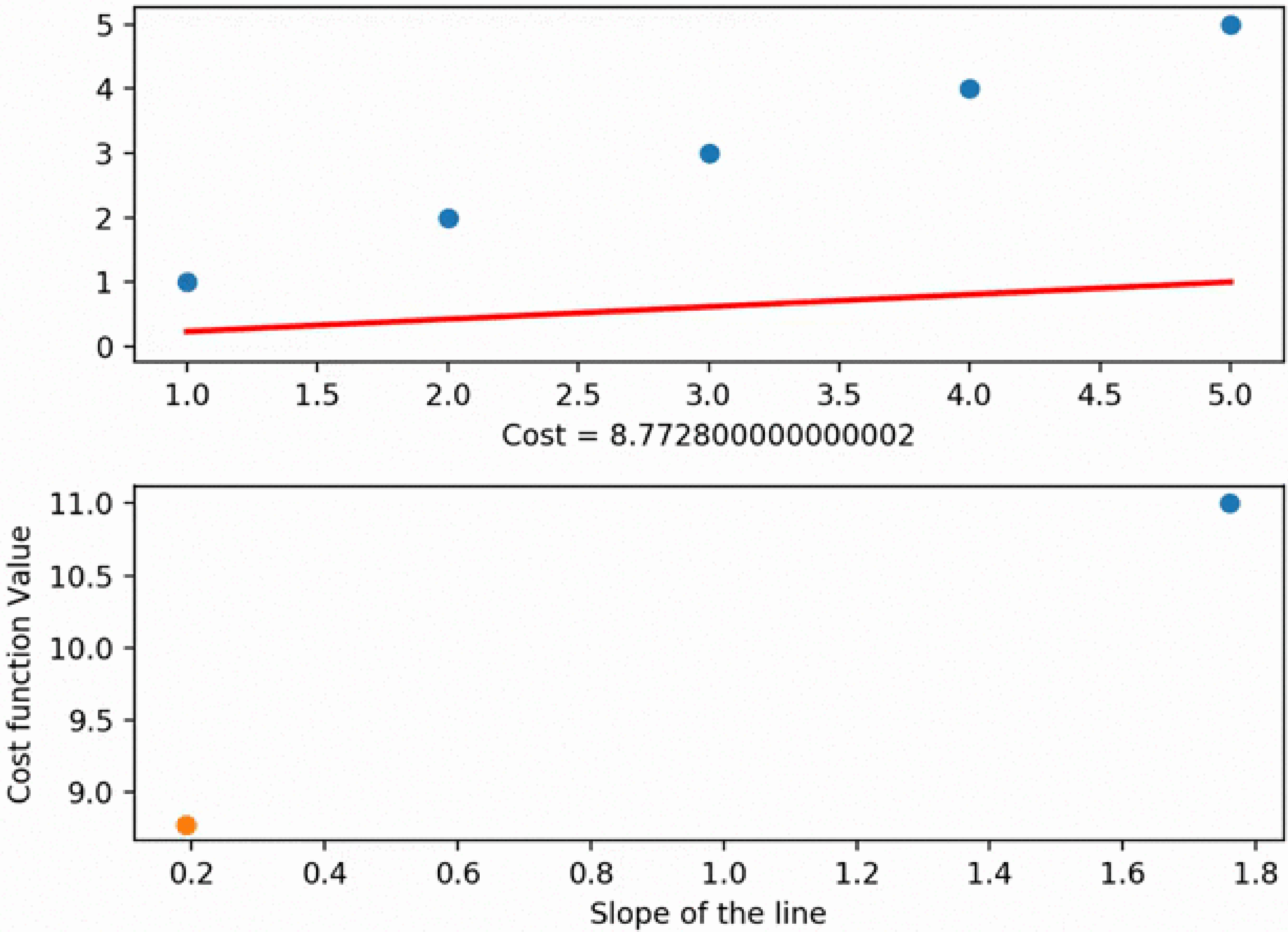
$y(x) = ax + b$ where a and b are unknown real values



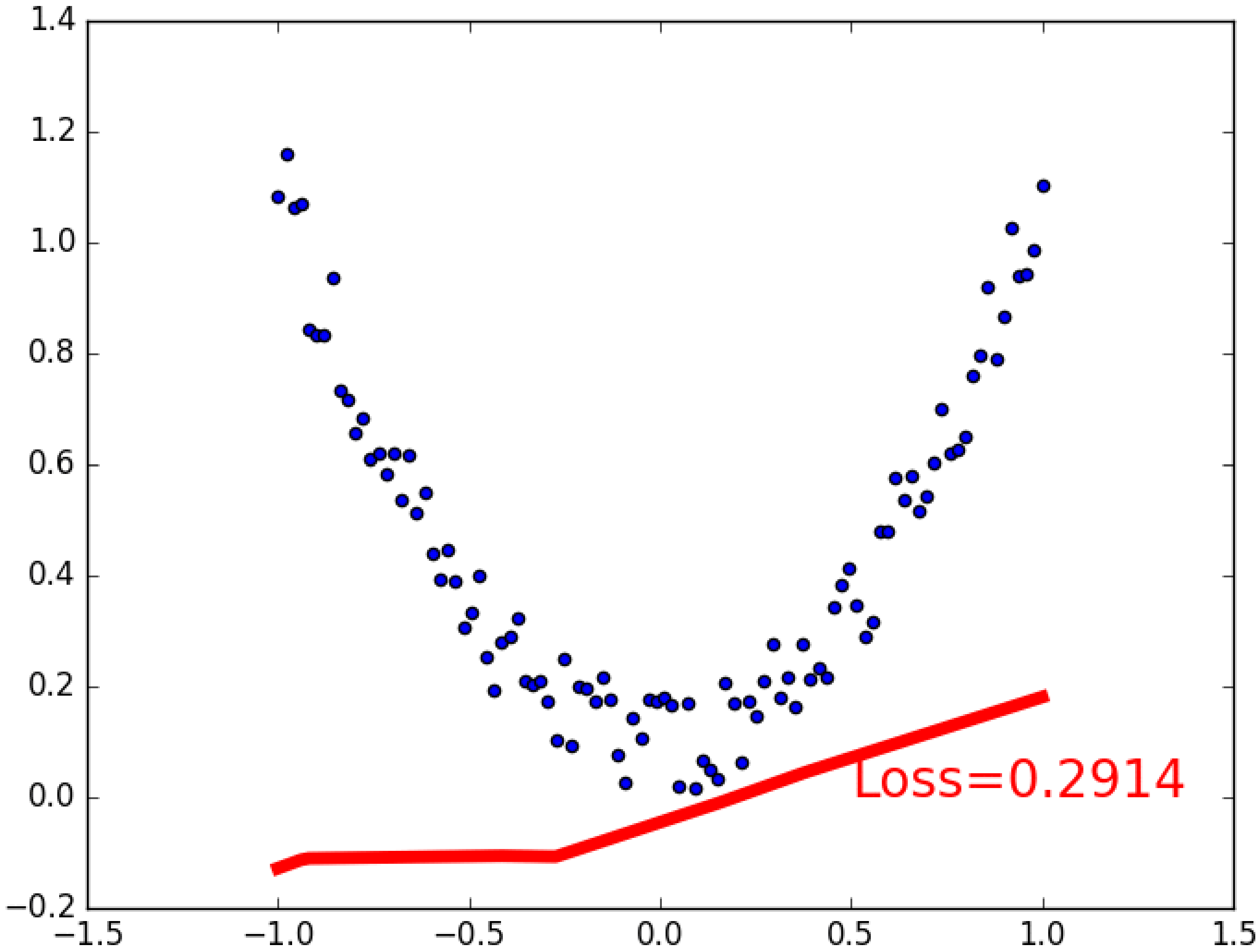


$y(x) = .8\ x + .4$

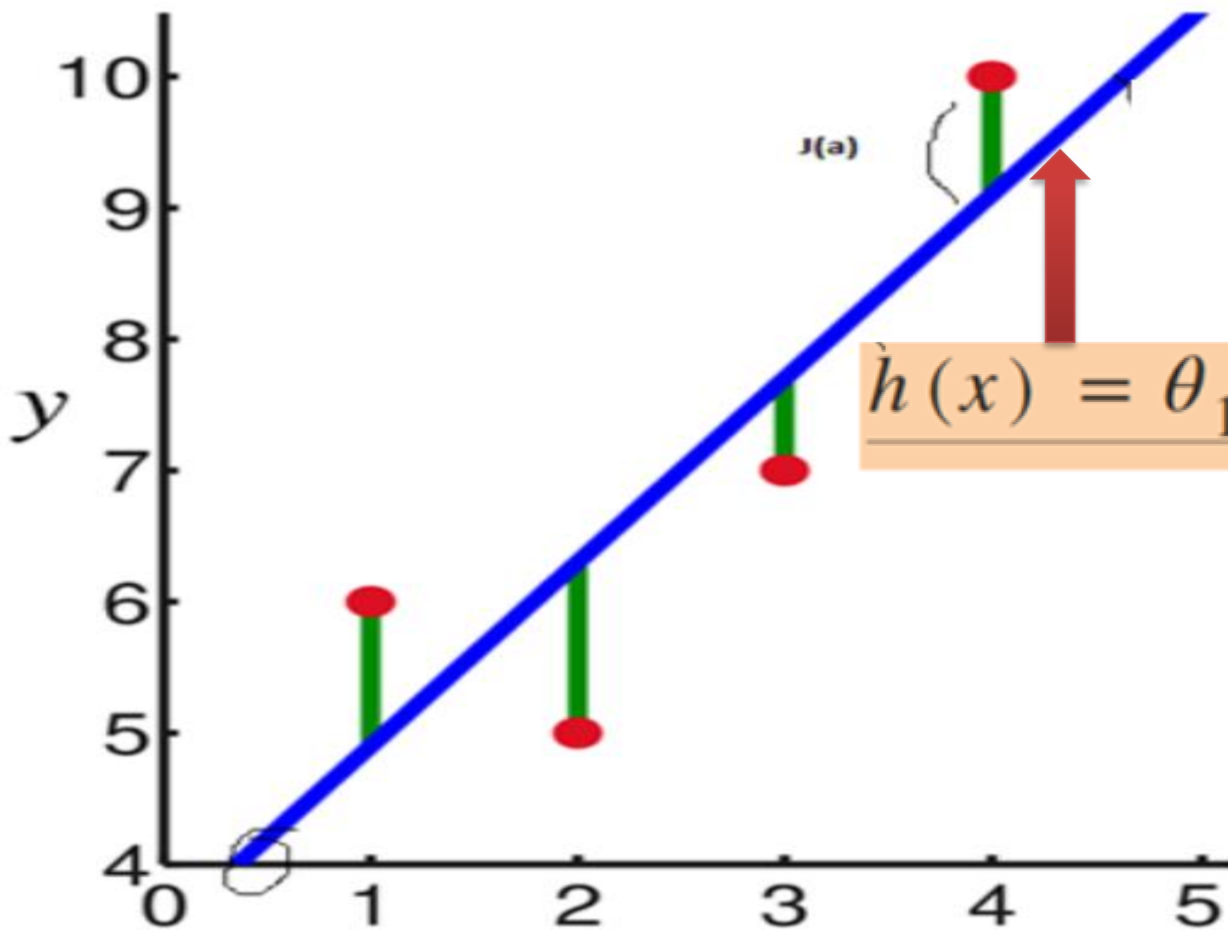
$y(x) = 1.3\ x + 2.5$



Cost = 8.772800000000002
Cost function Value
Slope of the line



1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Loss=0.2914



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

Cost Function

cost function is often the squared of the difference between actual and predicted outcome, because the difference can sometimes can be negative; this is also known as ***min least-squared***

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Error of the Model

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

$$RAE = \frac{\sum_{j=1}^{n}|y_j - \hat{y}_j|}{\sum_{j=1}^{n}|y_j - \bar{y}|}$$

$$RSE = \frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2}$$

$$\boxed{R^2 = 1 - RSE}$$

Problem

X (sleep, study)	y (test score)
(3,5)	75
(5,1)	82
(10,2)	93
(8,3)	?

we want to predict our test score based on how many hours we sleep and how many hours we study the night before

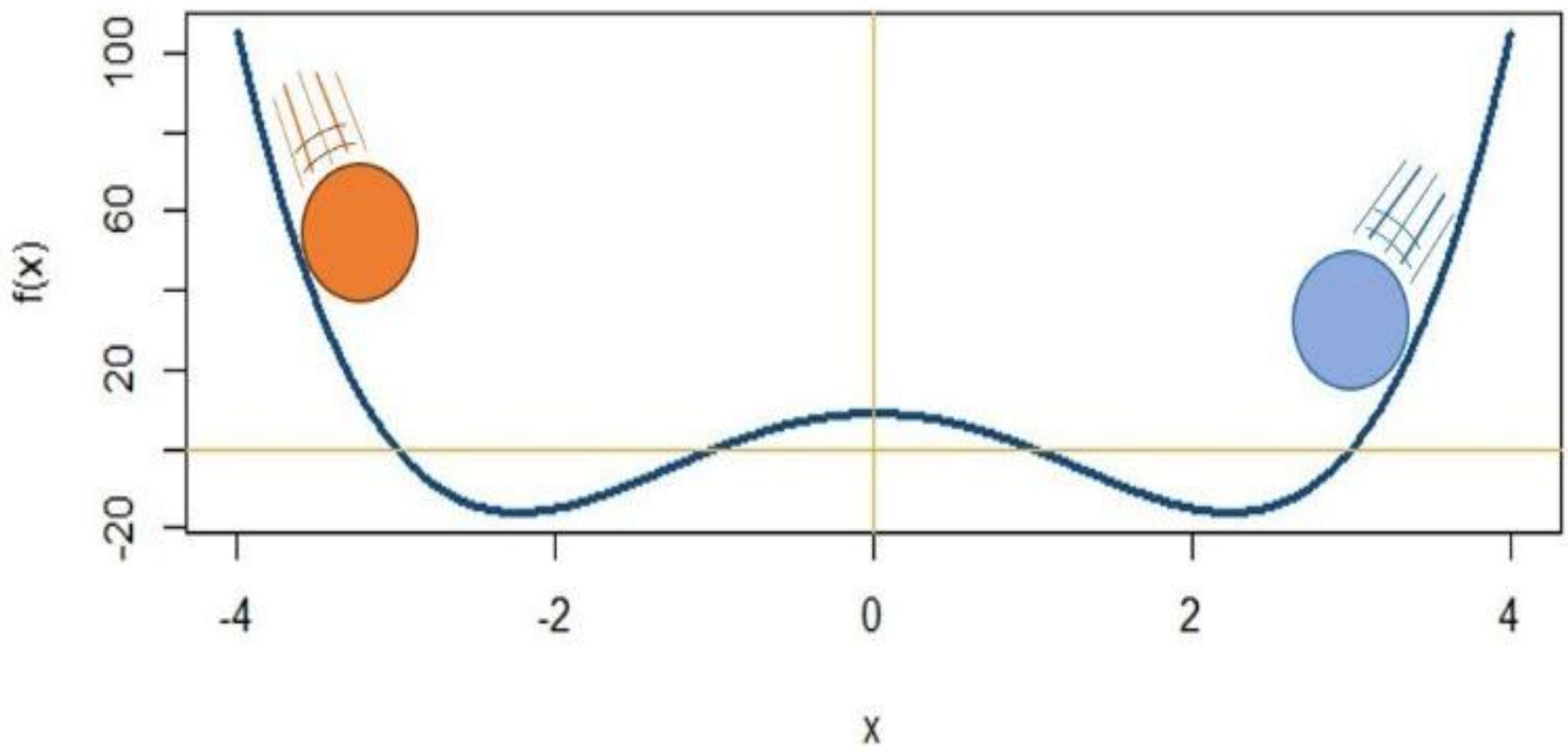
Finding Minimum Value of a Function

$$f(x) : x^4 - 10x^2 + 9$$

$$\frac{d}{dx} f(x) = 4x^3 - 20x = 0$$

$$Minima f(x) : x = \pm\sqrt{5} = \pm 2.24$$

Iterative Calculation



Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

$$\text{Error}_{(m,b)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

$$\frac{\partial}{\partial \text{m}} = \frac{2}{N}\sum_{i=1}^{N} -x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial \text{b}} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (mx_i + b))$$

Gradient Descent

$$Loss\ Function\ (LF) = \frac{1}{N}\sum(y - (mx + c))^2$$

$$\frac{\partial}{\partial m}LF = \frac{2}{N}\sum(y - (mx + c)) \times x$$

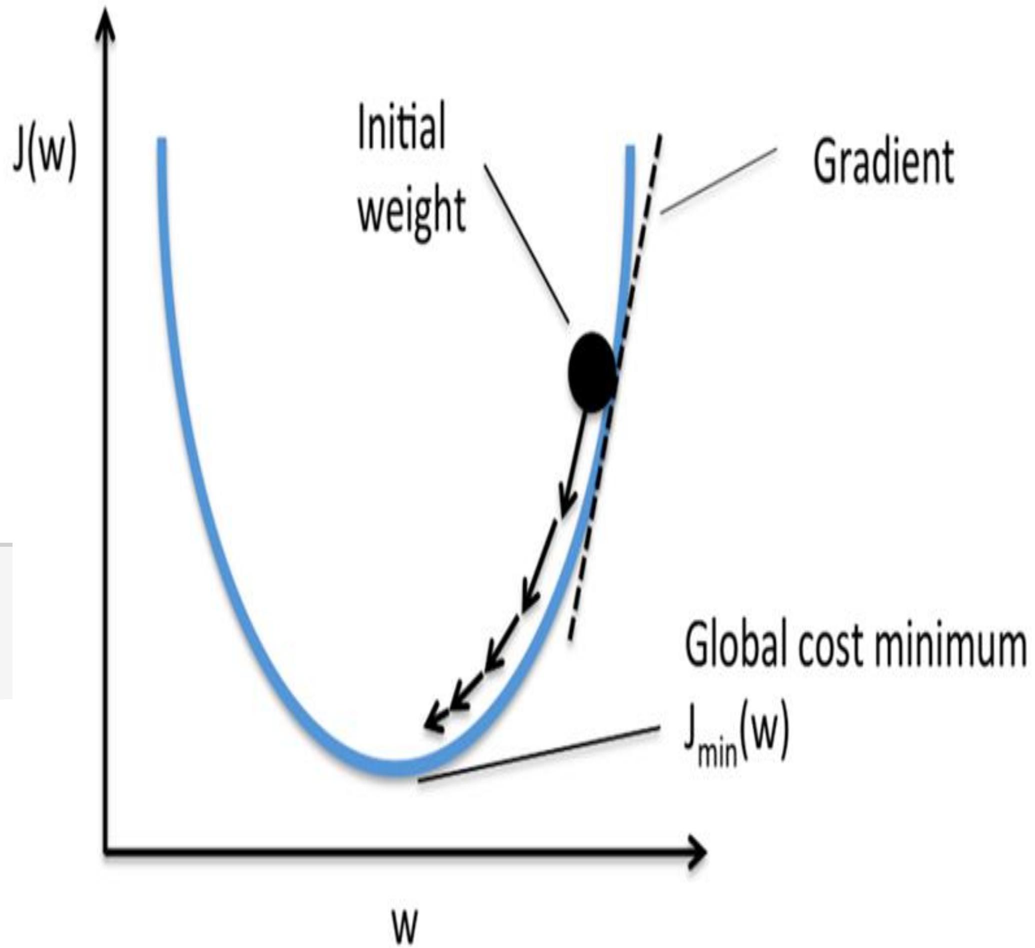
$$\frac{\partial}{\partial c}LF = \frac{2}{N}\sum(y - (mx + c))$$

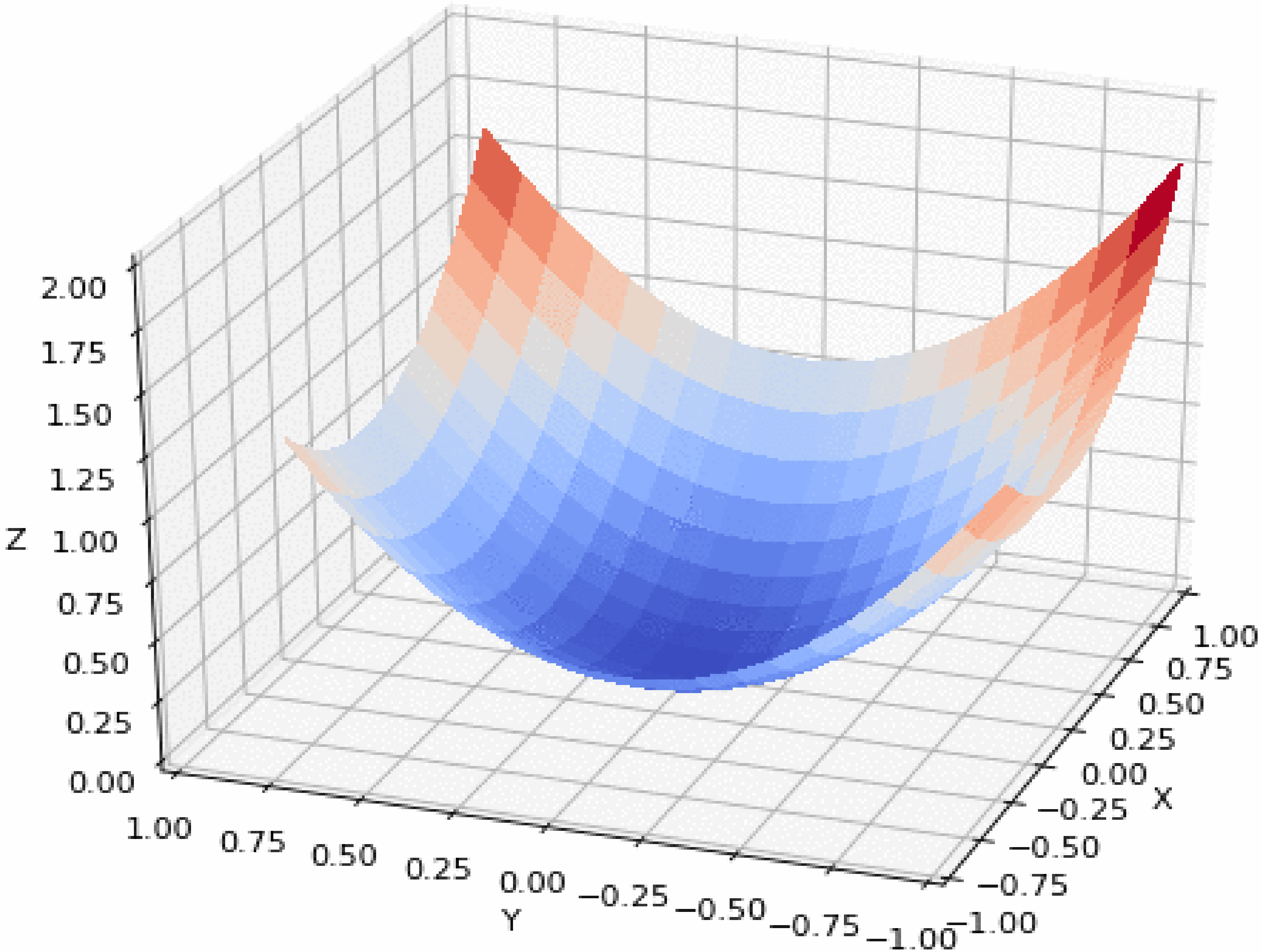
when our model coefficients are trained. During training, our initial weights are updated according to a gradient update rule using a learning rate and a gradient

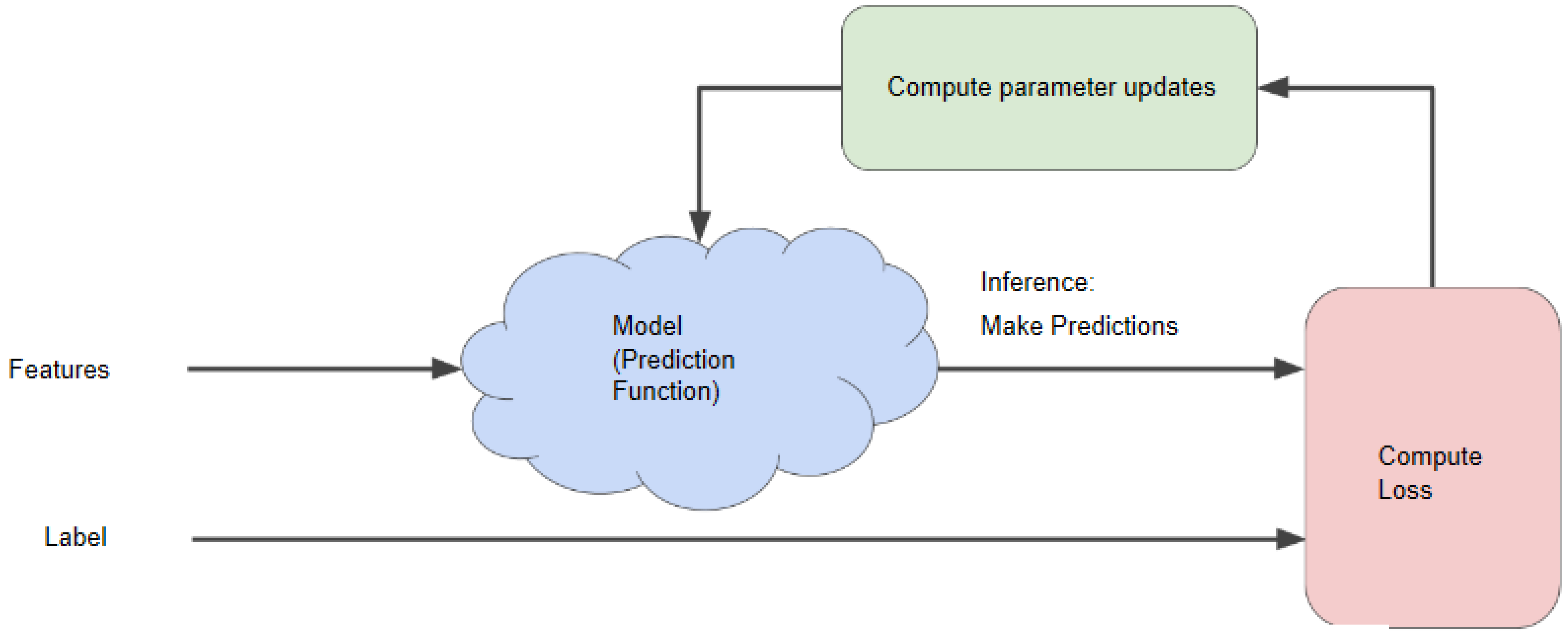
Iterative Calculation

Repeat until converge

$$\left\{ \begin{aligned} m_{n+1} &= m_n - \alpha \frac{\partial}{\partial m_n} LF(m_n) \\ c_{n+1} &= c_n - \alpha \frac{\partial}{\partial c_n} LF(c_n) \end{aligned} \right\}$$




2.00
1.75
1.50
1.25
Z
1.00
0.75
0.50
0.25
0.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
X

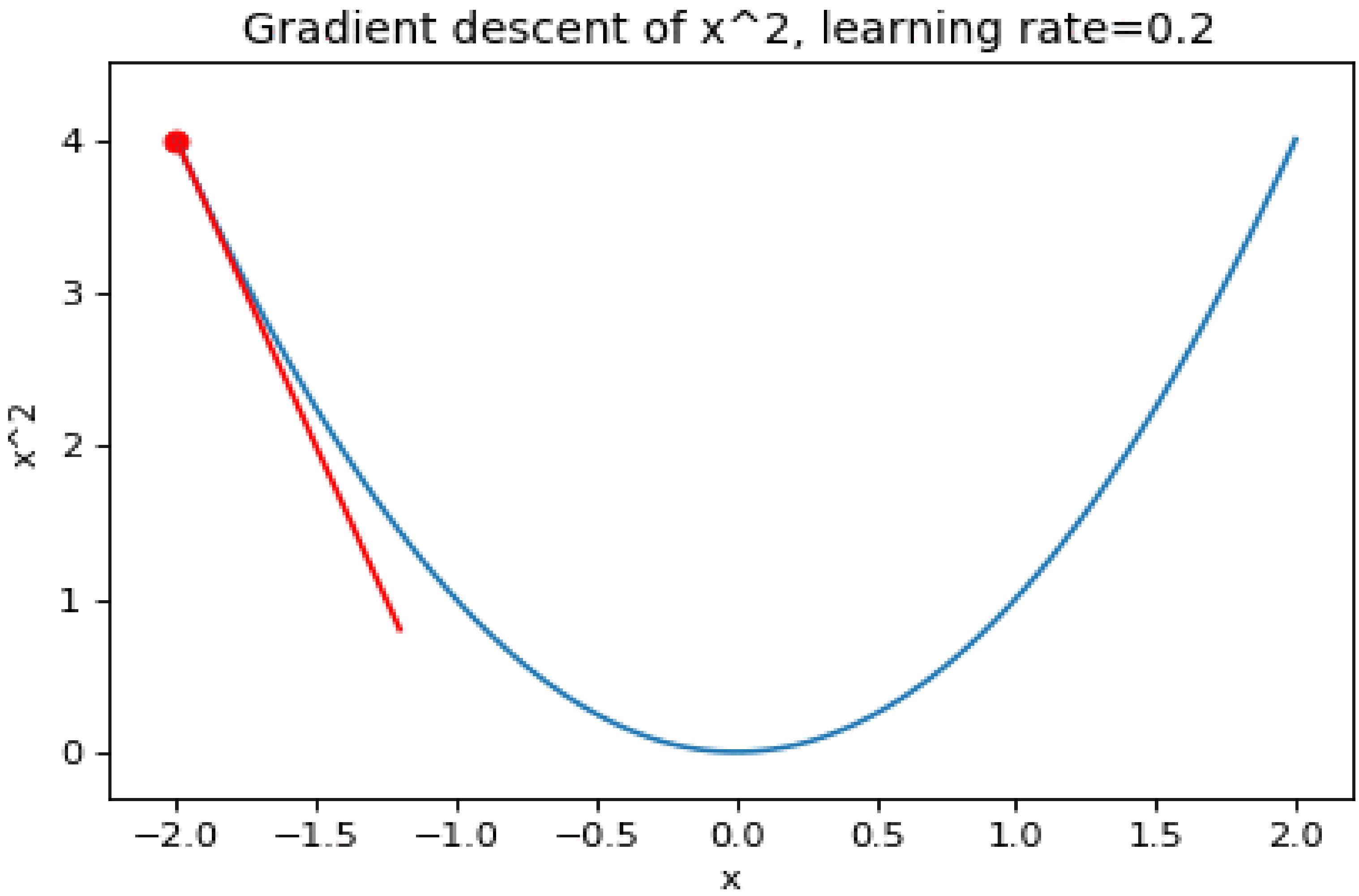


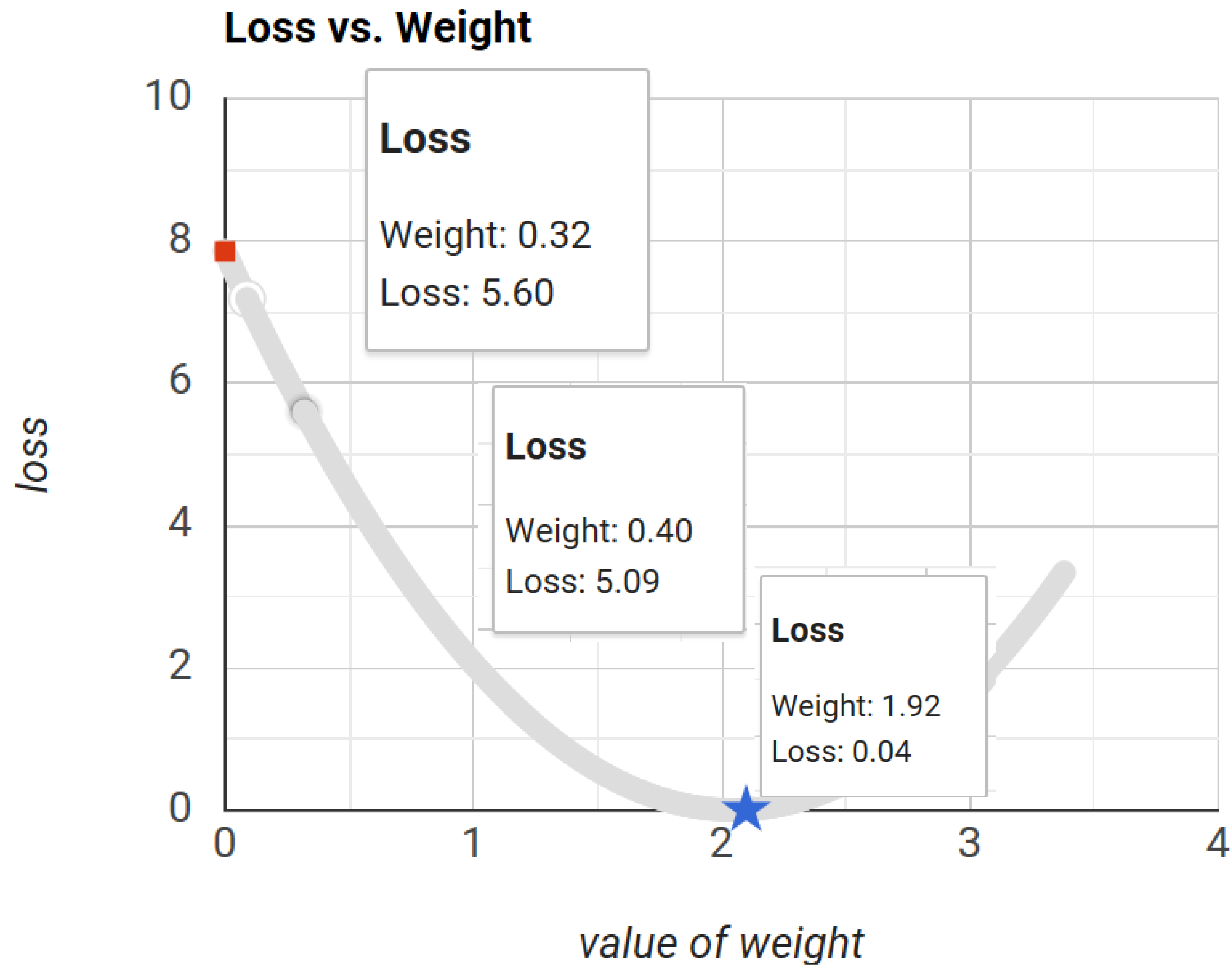
repeat until convergence {

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

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

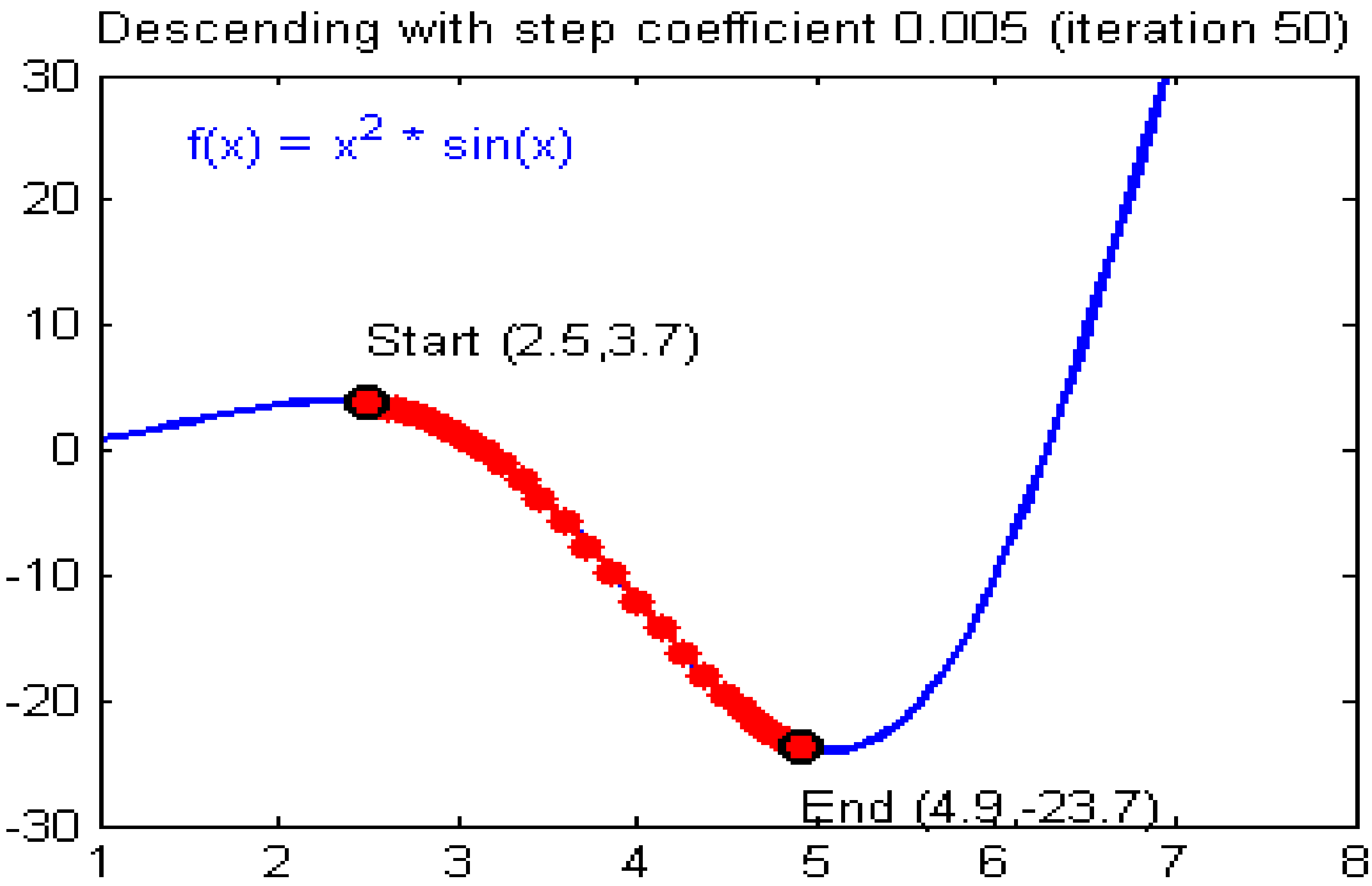
}

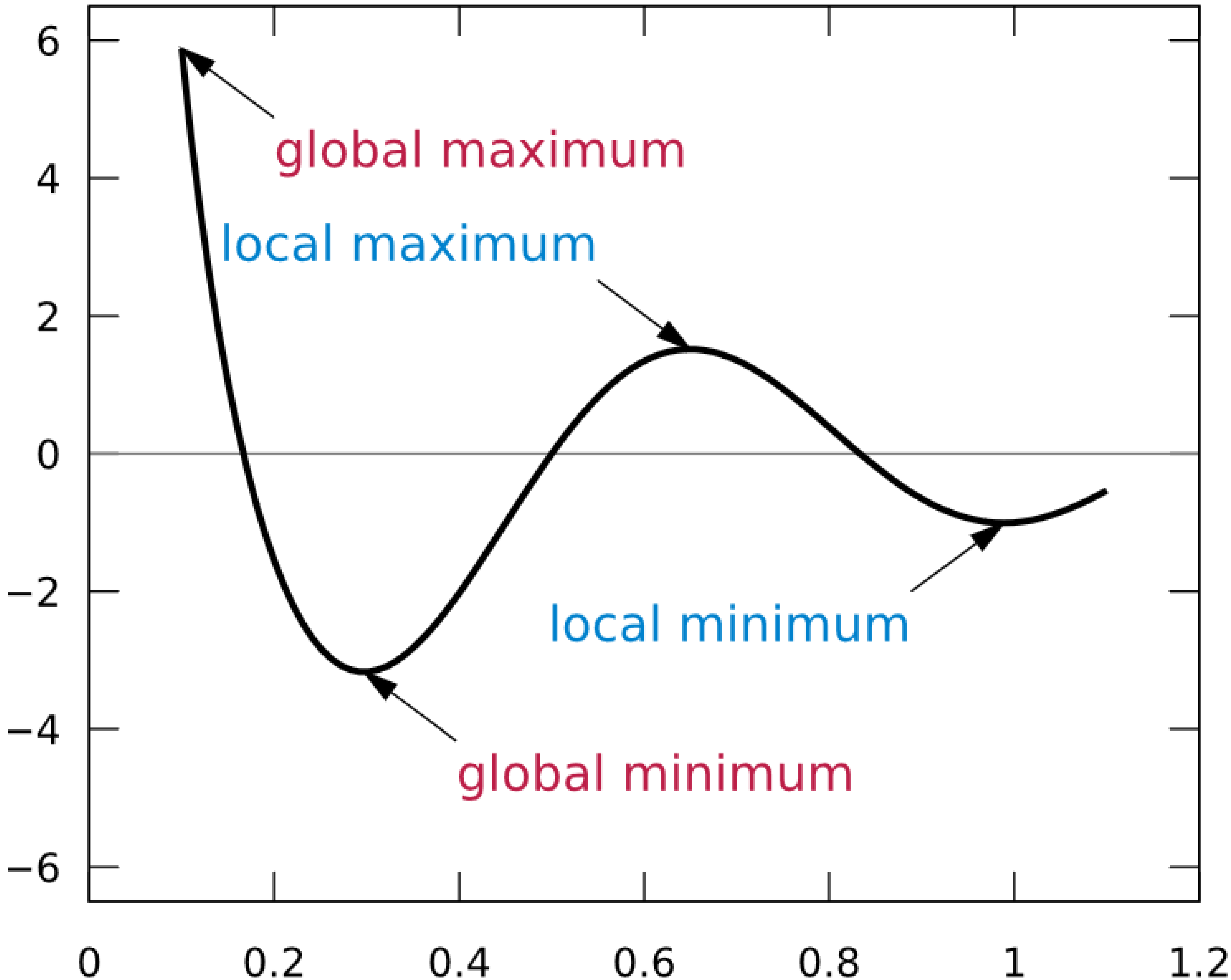

Gradient descent of x^2, learning rate=0.2
4
3
2
1
0
x^2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x



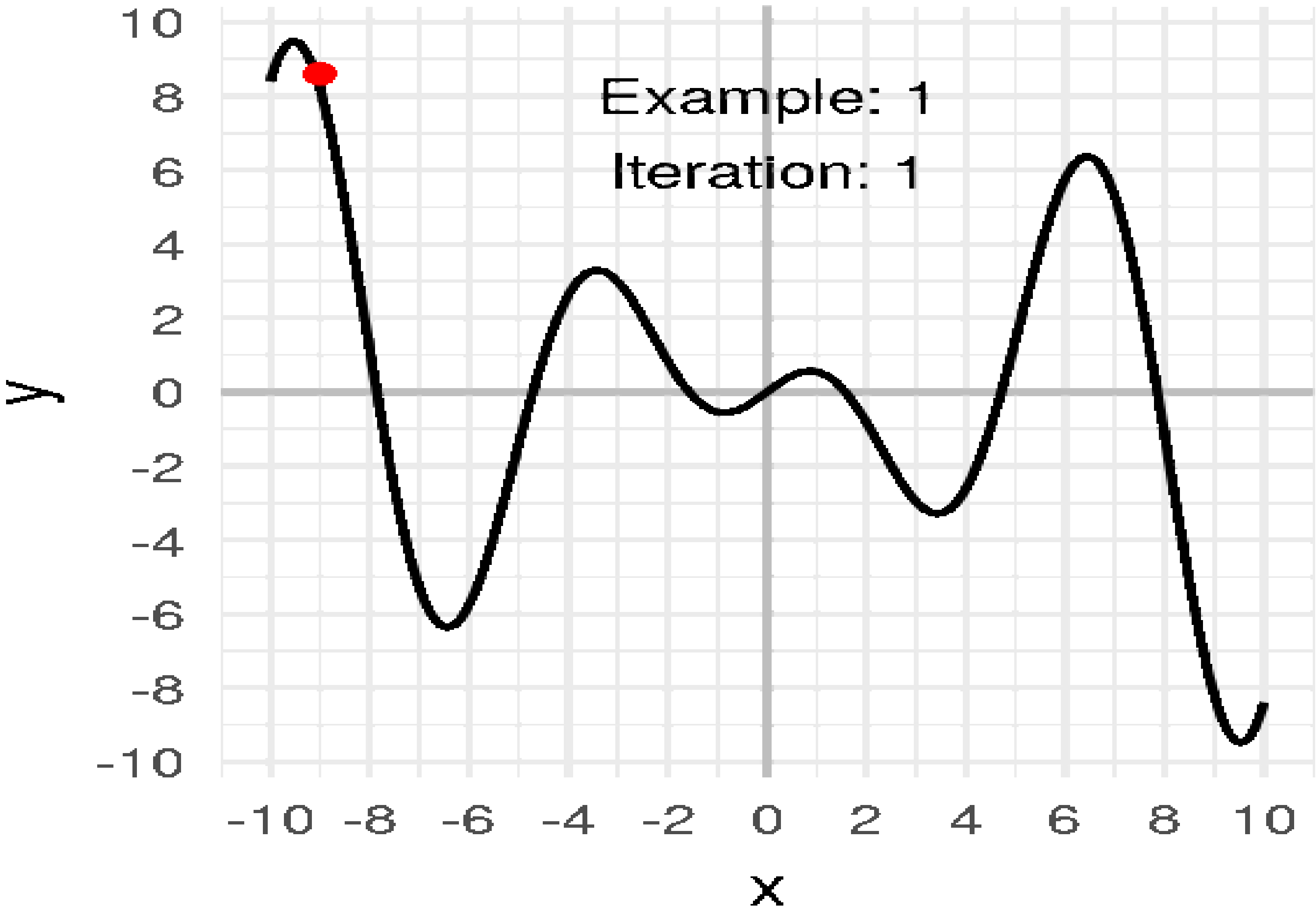
www.chandanverma.com

Gradient Descent





www.chandanverma.com


Example: 1
Iteration: 1
10
8
6
4
2
0
-2
-4
-6
-8
-10
y
-10
-8
-6
-4
-2
0
2
4
6
8
10
x

X_INPUT	Y_OUTPUT
1	6
2	5
3	7
4	10

$$y = \beta_1 + \beta_2 x$$

$$\begin{aligned} \beta_1 + 1\beta_2 &= 6 \\ \beta_1 + 2\beta_2 &= 5 \\ \beta_1 + 3\beta_2 &= 7 \\ \beta_1 + 4\beta_2 &= 10 \end{aligned}$$

$$\begin{aligned} S(\beta_1, \beta_2) =& [6 - (\beta_1 + 1\beta_2)]^2 + [5 - (\beta_1 + 2\beta_2)]^2 \\ & + [7 - (\beta_1 + 3\beta_2)]^2 + [10 - (\beta_1 + 4\beta_2)]^2 \\ & = 4\beta_1^2 + 30\beta_2^2 + 20\beta_1\beta_2 - 56\beta_1 - 154\beta_2 + 210. \end{aligned}$$

The minimum is determined by calculating the partial derivatives of S (β 1 , β 2) with respect to β 1 and β 2 and setting them to zero

$$\frac{\partial S}{\partial \beta_1} = 0 = 8\beta_1 + 20\beta_2 - 56$$
$$\frac{\partial S}{\partial \beta_2} = 0 = 20\beta_1 + 60\beta_2 - 154.$$

$$\beta_1 = 3.5$$
$$\beta_2 = 1.4$$

equation y = 3.5 + 1.4 x of the line of best fit

Population Y intercept

Population Slope Coefficient

Independent Variable

Random Error term

Dependent Variable

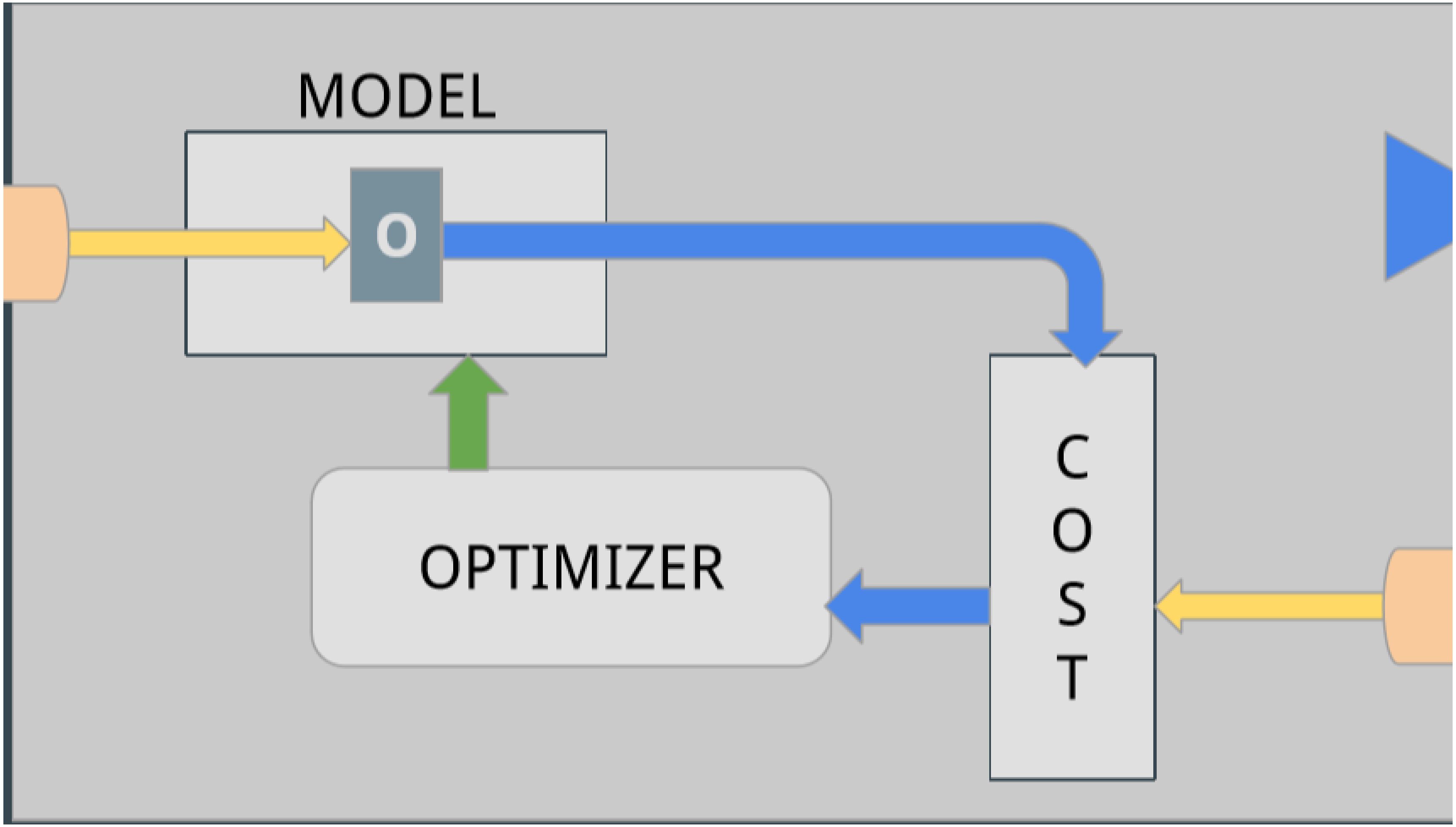
$$Y_i = \underbrace{\beta_0 + \beta_1 X_i}_{\text{Linear component}} + \underbrace{\varepsilon_i}_{\text{Random Error component}}$$

Regression-model

$$\underbrace{\boxed{\hat{y}}}_{\text{Predicted output}} = \underbrace{\beta_0 \;+\; \beta_1}_{\text{Coefficients}} \underbrace{\boxed{x}}_{\text{Input}} + \underbrace{\boxed{\epsilon}}_{\text{Error}}$$

$$\boxed{\hat{y}} = \underbrace{\beta_0 \;+\; \beta_1}\boxed{x_1} + \dots + \underbrace{\beta_p}\boxed{x_p} + \boxed{\epsilon}$$

Regression



Cost (loss) function

$$x \cdot W + b = y(\text{target})$$

prediction

$$y - (x \cdot W + b)$$

$$[\, y - (x \cdot W + b)\,]^2$$

$$\Sigma[\, yi - (xi \cdot W + b)\,]^2$$

```
cost = tf.reduce_sum(tf.square(output - y))
```

Assumptions of linear regression

1. There must be a linear relationship between the dependent and independent variables.
2. Error terms are normally distributed with mean 0.
3. No multicollinearity - When the independent variables in my model are highly linearly related then such a situation is called multicollinearity.
4. No outliers are present in the data.

sklearn.linear_model.LinearRegression

- fit_intercept=True,
- normalize=False,
- copy_X=True,
- n_jobs=None)

variance, r2 score, and **mean square error**

It is important to understand these metrics to determine whether regression models are accurate or misleading.

variance—in terms of linear regression, **variance** Is a measure of how far observed values differ from the average of predicted values, i.e., their difference from the **predicted value mean**.

r2 score—varies between 0 and 100%. It is closely related to the **MSE** , but not the same

R square

It suggests the proportion of variation in Y which can be explained with the independent variables. Mathematically, it is the ratio of predicted values and observed values

R-Squared is a statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variable(s) in a regression model.

$$R^2 = 1 - \frac{Explained\ Variation}{Total\ Variation}$$

R-squared can take value between 0 and 1 where values closer to 0 represent a poor fit while values closer to 1 represent a perfect fit

$$R^2 = \frac{SSR}{SST} = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

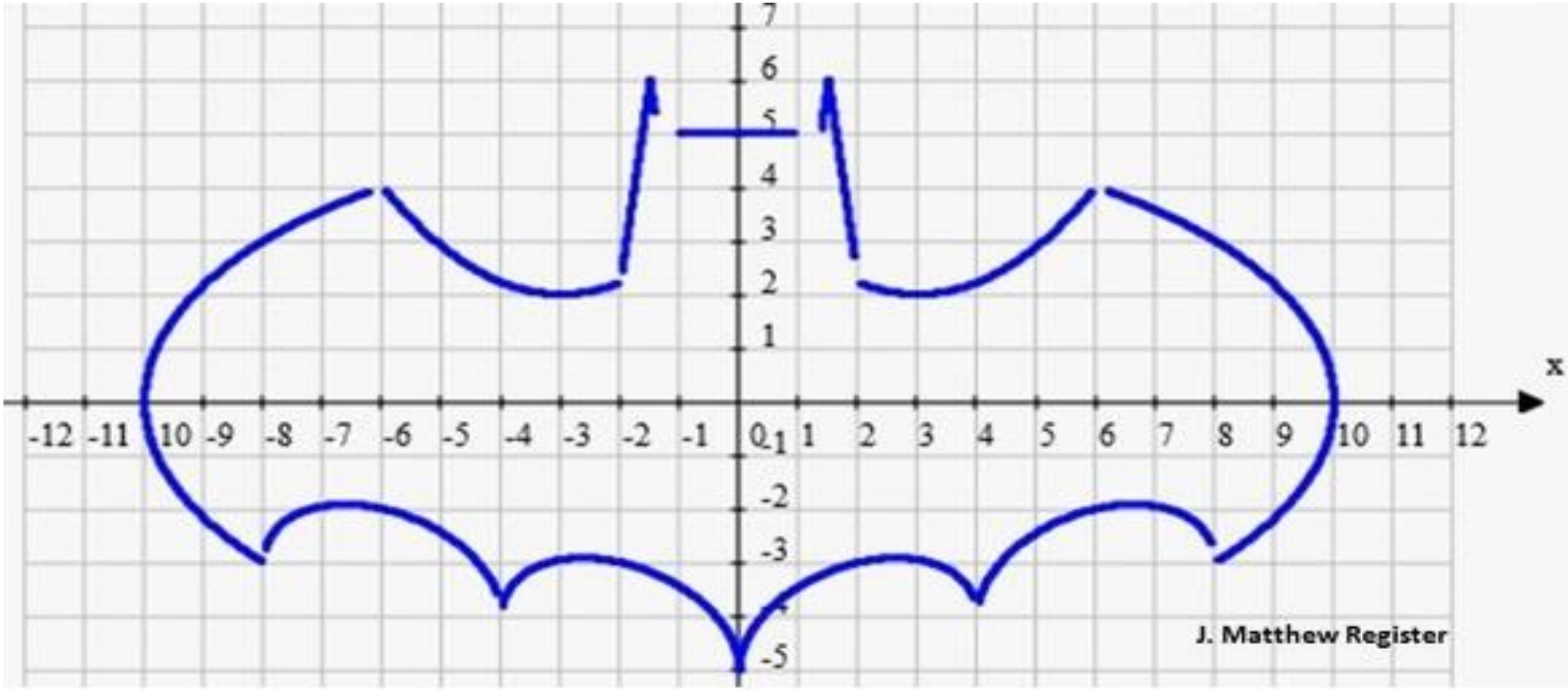
$$R - Square = 1 - \frac{\Sigma(Y_{actual} - Y_{predicted})^2}{\Sigma(Y_{actual} - Y_{mean})^2}$$

$$R^2 = \frac{\sum y_i^2 - \sum e_i^2}{\sum y_i^2} = 1 - \frac{\sum e_i^2}{\sum y_i^2}$$

Adjusted R-square

The only drawback of R^2 is that if new predictors (X) are added to our model, R^2 only increases or remains constant but it never decreases. We can not judge that by increasing complexity of our model, are we making it more accurate?

$$R^2 \text{ adjusted} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$



$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 \cdots$$

gradientdescentoptimizer

Gradient Descent is a learning algorithm that attempts to minimise some error

performing gradient descent to minimize the cost function, to obtain the ‘good’ values for W, b.

```
optimizer =
tf.train.GradientDescentOptimizer(learning_rate=0.00001)

optimizer = optimizer.minimize(cost)
```

The learning rate is how quickly a network **abandons** old beliefs for new ones

Linear Regression Exercise

- Creating training data
- Placeholders
- Modeling
- Training

Linear Model (in TF notation):

y = tf.matmul(x,W) + b

The goal in linear regression is to find W, b, such that given any feature value (x), we can find the prediction (y) by substituting W, x, b values into the model.

need to define a cost function, which is a measure of the ***difference*** between the **prediction** (y) for given a feature value (x), and the **actual outcome** (y_) for that same feature value (x).

Model linear regression y = Wx + b

$$[\ y - (\ x \cdot W + b\ )\ ]^2$$

tf.train.GradientDescentOptimizer(LR).minimize(cost)

```
cost = tf.reduce_mean(tf.pow((y_-y), 2))

y = tf.matmul(x,W) + b

y_ = tf.placeholder(tf.float32, [None, 1])

x = tf.placeholder(tf.float32, [None, 1])
```

```
import numpy as np
import tensorflow as tf
x = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.zeros([1,1]))
b = tf.Variable(tf.zeros([1]))
product = tf.matmul(x,W)
y = product + b
y_ = tf.placeholder(tf.float32, [None, 1])
cost = tf.reduce_mean(tf.square(y_-y))
train_step =
tf.train.GradientDescentOptimizer(0.0000001).minimize(cost)
```

```
sess = tf.Session()
init = tf.initialize_all_variables()
sess.run(init)
steps = 1000
for i in range(steps):
xs = np.array([[i]])
ys = np.array([[2*i]])
feed = { x: xs, y_: ys }
sess.run(train_step, feed_dict=feed)
 print("After %d iteration:" % i)
 print("W: %f" % sess.run(W))
  print("b: %f" % sess.run(b))
  print("cost: %f" % sess.run(cost, feed_dict=feed))
```