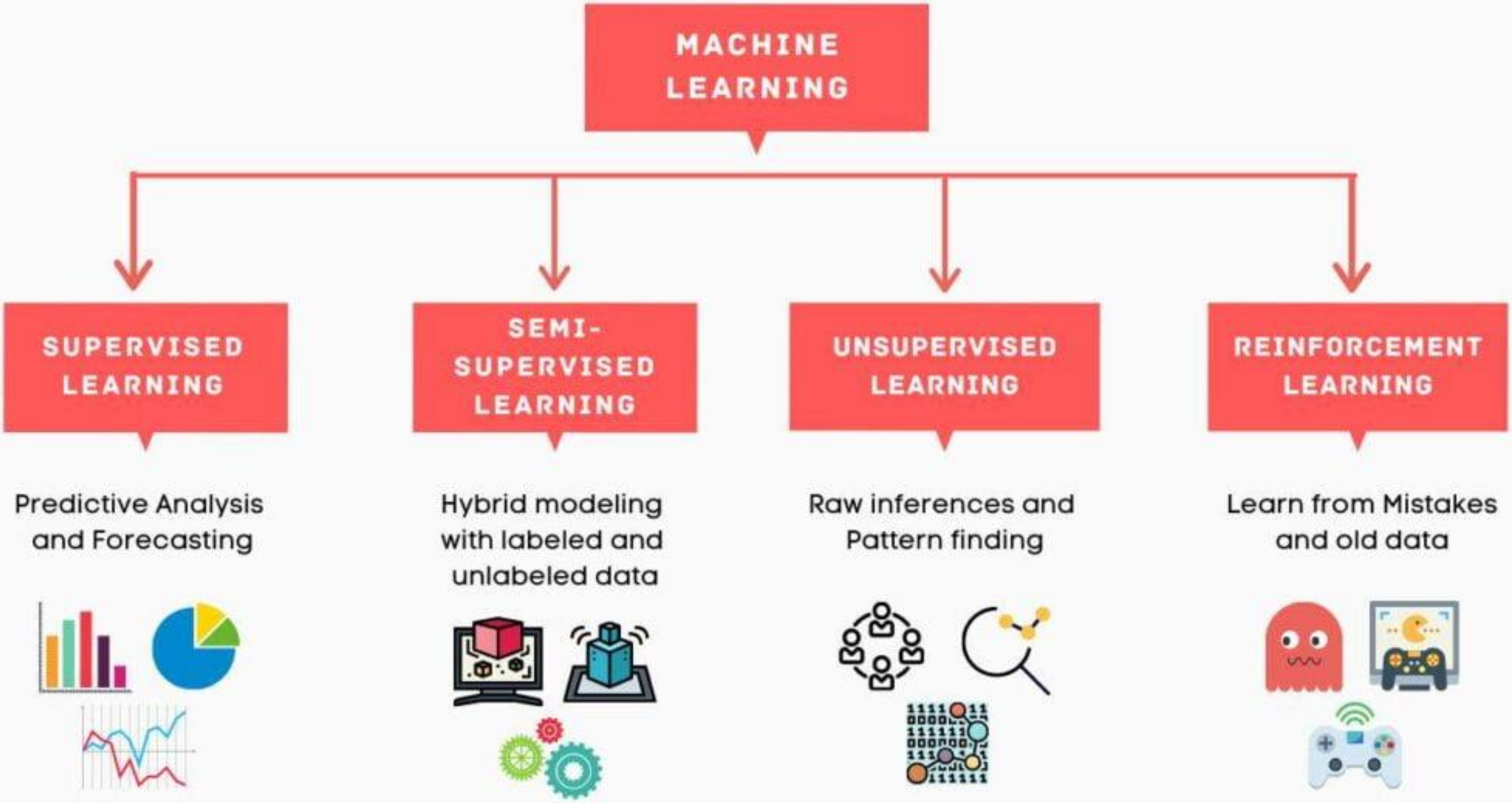
AI VIET NAM
@aivietnam.edu.vn

Linear Regression (Review and Assignments)

Vinh Dinh Nguyen
PhD in Computer Science

Outline

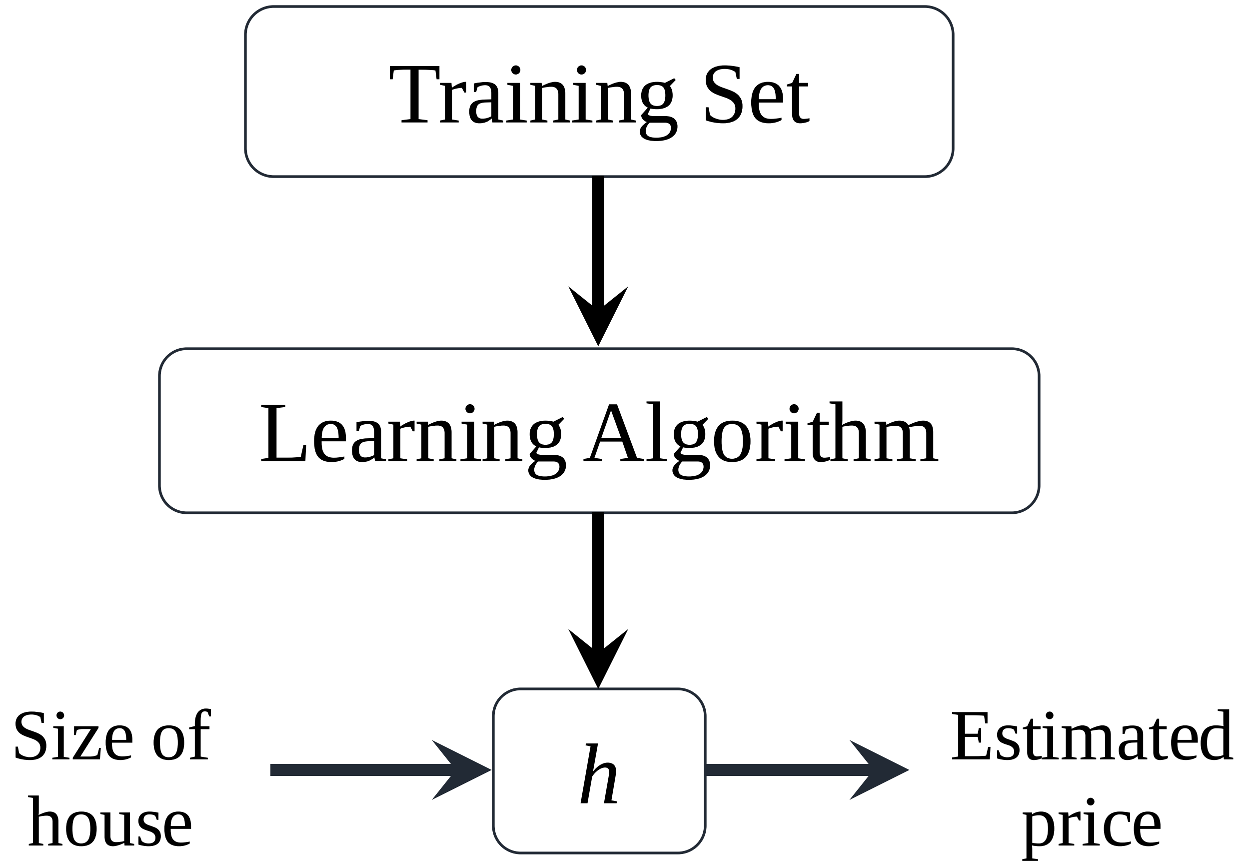
- Linear Regression Review
- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
- Other Discussions


MACHINE LEARNING
SUPERVISED LEARNING
SEMI-SUPERVISED LEARNING
UNSUPERVISED LEARNING
REINFORCEMENT LEARNING
Predictive Analysis and Forecasting
Hybrid modeling with labeled and unlabeled data
Raw inferences and Pattern finding
Learn from Mistakes and old data

Linear Regression: Quick Review

Area (feet^2)	Price ($)
2140	460k
1416	232k
1534	315k
852	178k
500	135k

Linear Regression: Quick Review



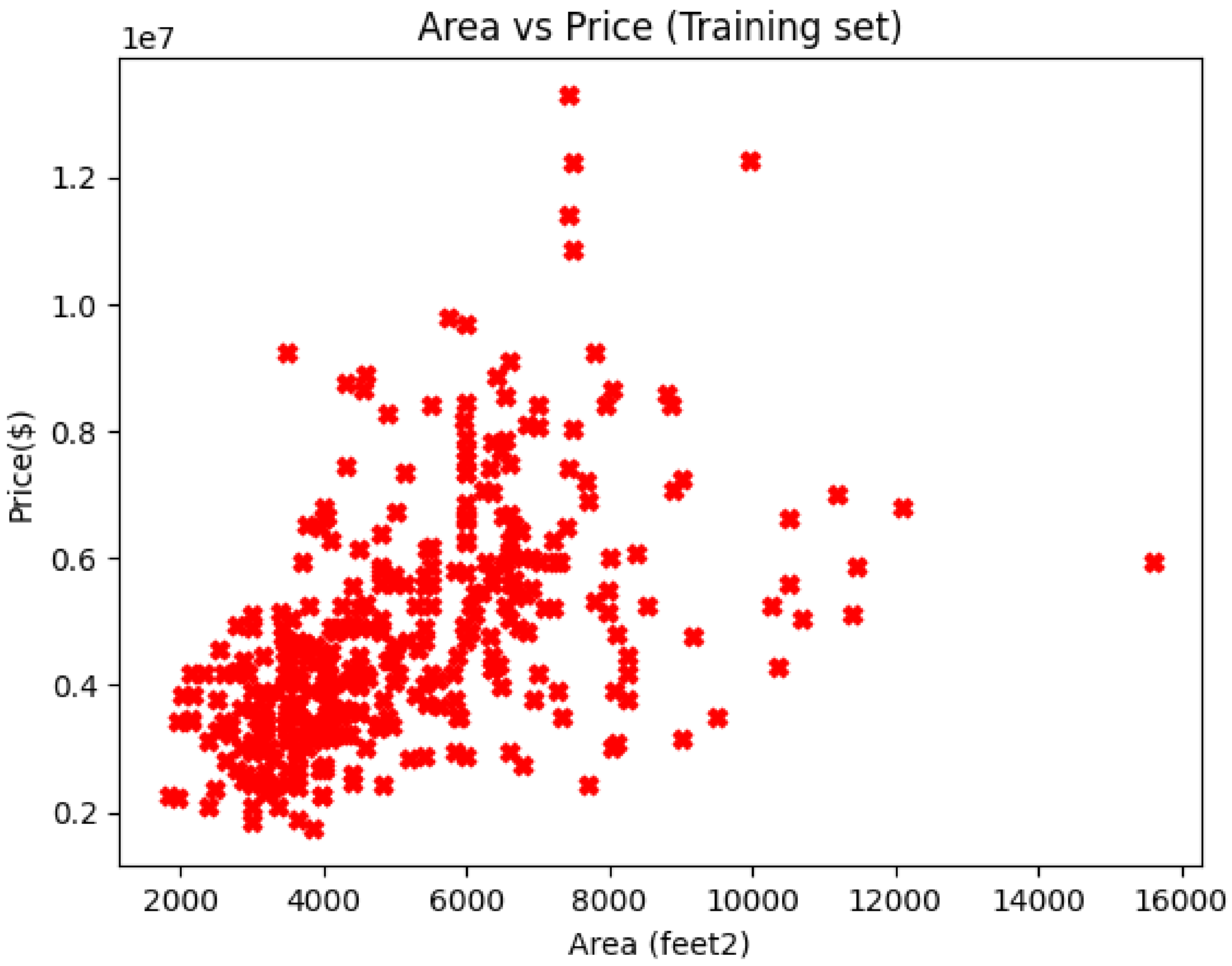
How do we represent *h* ?

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

Linear Regression with One Variable
Univariate linear regression

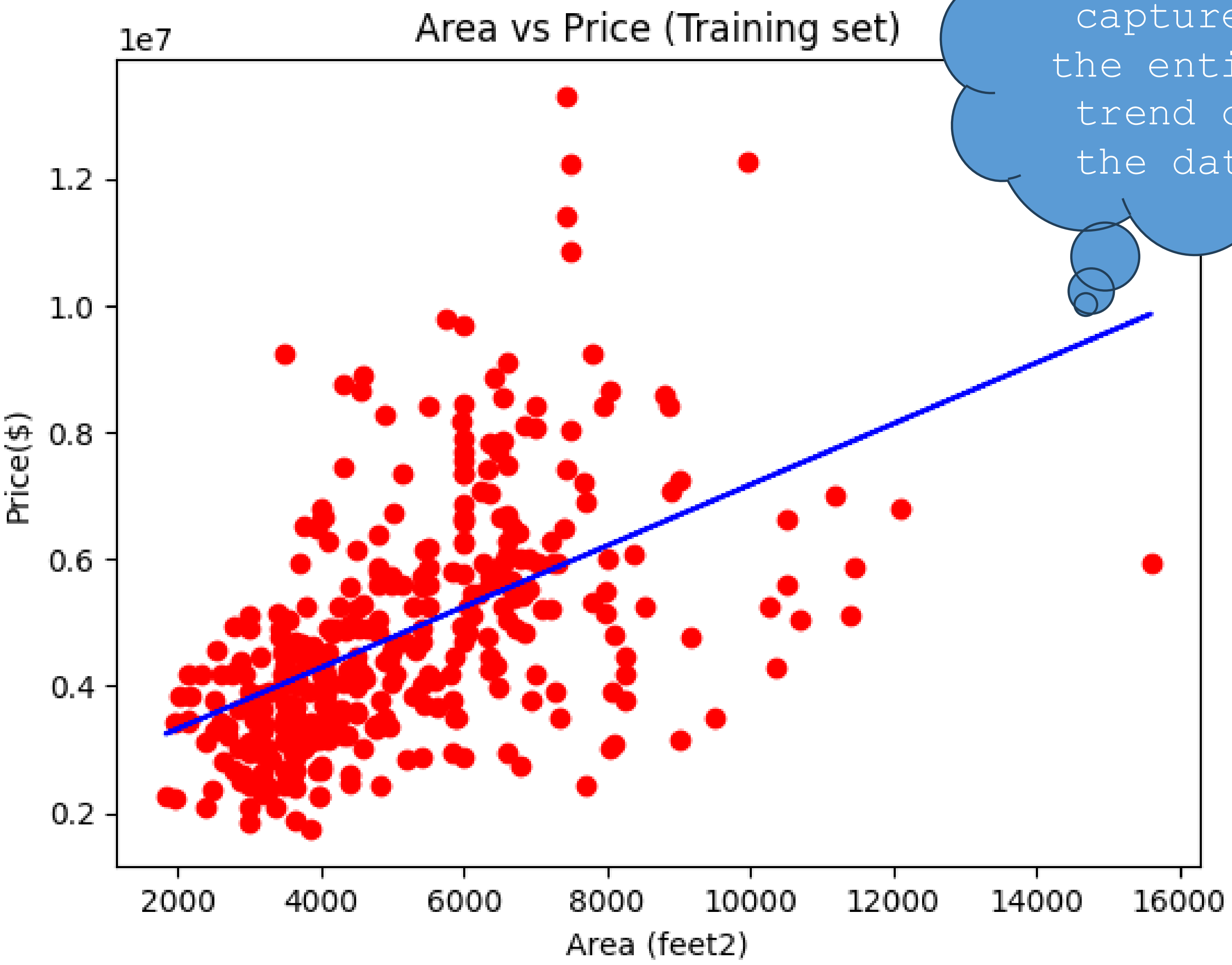
AI VIET NAM
@aivietnam.edu.vn

Linear Regression: Quick Review



AI VIET NAM
@aivietnam.edu.vn

Linear Regression



Can we find a line that captures the entire trend of the data

Linear Regression: Quick Review

Training Set

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

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

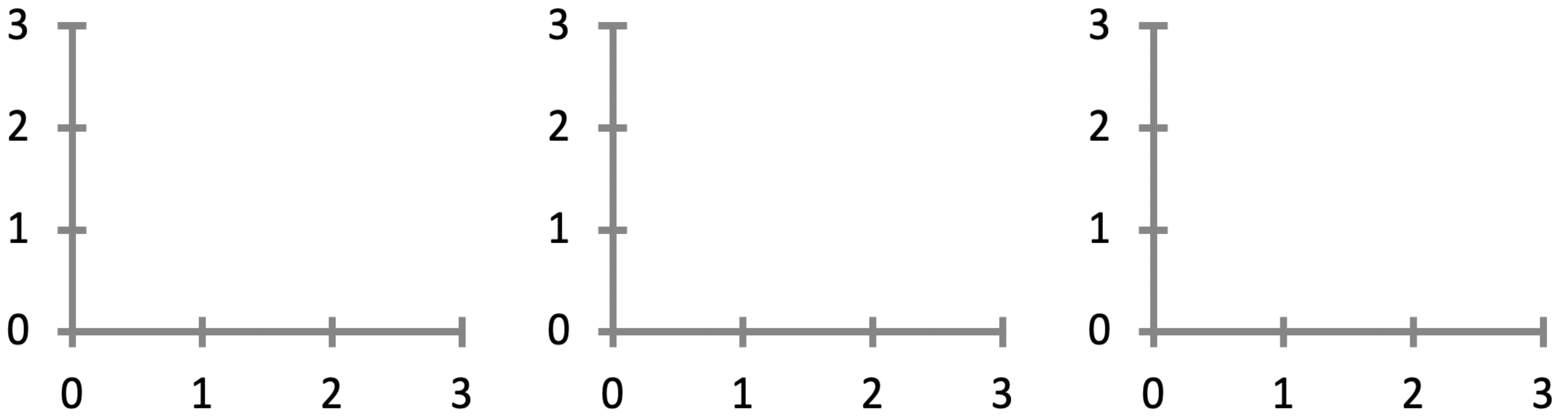
θ_i's: Parameters

How to choose θ_i's ?

AI VIET NAM
@aivietnam.edu.vn

Linear Regression: Quick Review

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

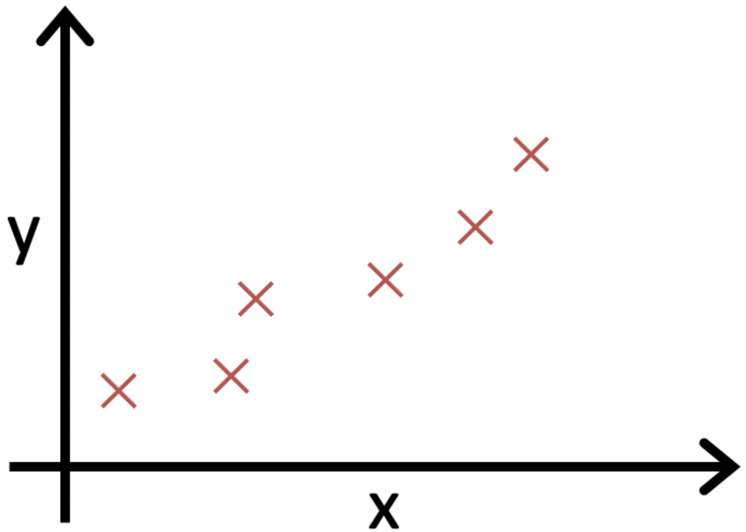


$$\theta_0 = 1.5 \quad \theta_1 = 0$$

$$\theta_0 = 0 \quad \theta_1 = 0.5$$

$$\theta_0 = 1 \quad \theta_1 = 0.5$$

Linear Regression: Quick Review



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

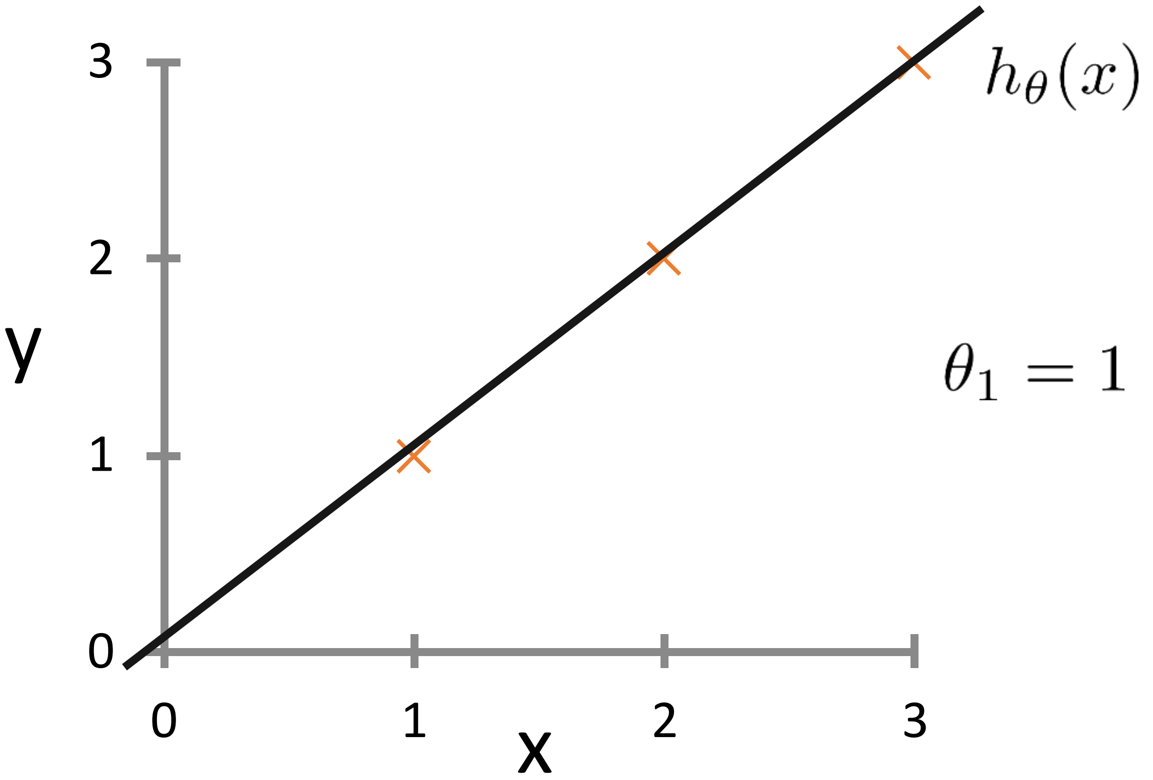
Idea: Choose θ_0, θ_1 so that
$h_\theta(x)$ is close to y for our
training examples (x, y)

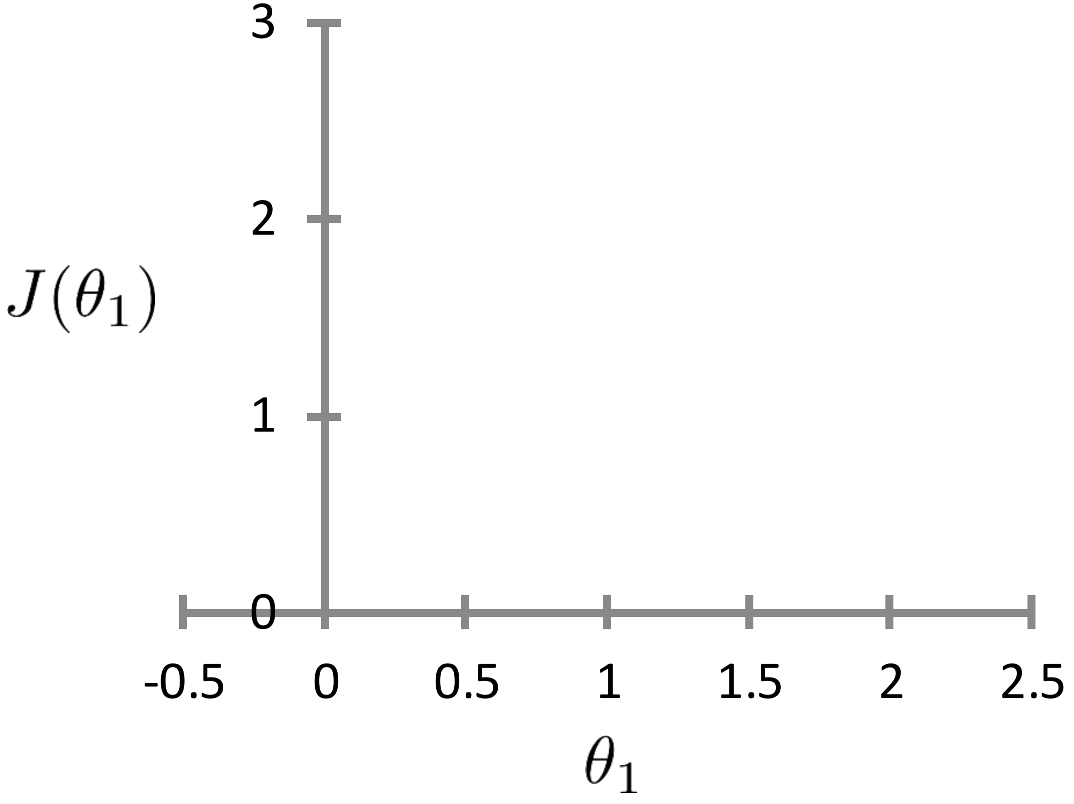
Linear Regression: Quick Review

	Simplified
Hypothesis: $$h_\theta(x) = \theta_0 + \theta_1 x$$	$$h_\theta(x) = \theta_1 x$$
Parameters: $$\theta_0, \theta_1$$	$$\theta_1$$
Cost Function: $$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$	$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$
Goal: $$\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$$	$$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$$

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

$h_\theta(x)$	$J(\theta_1)$
(for fixed θ_1, this is a function of x)	(function of the parameter θ_1)

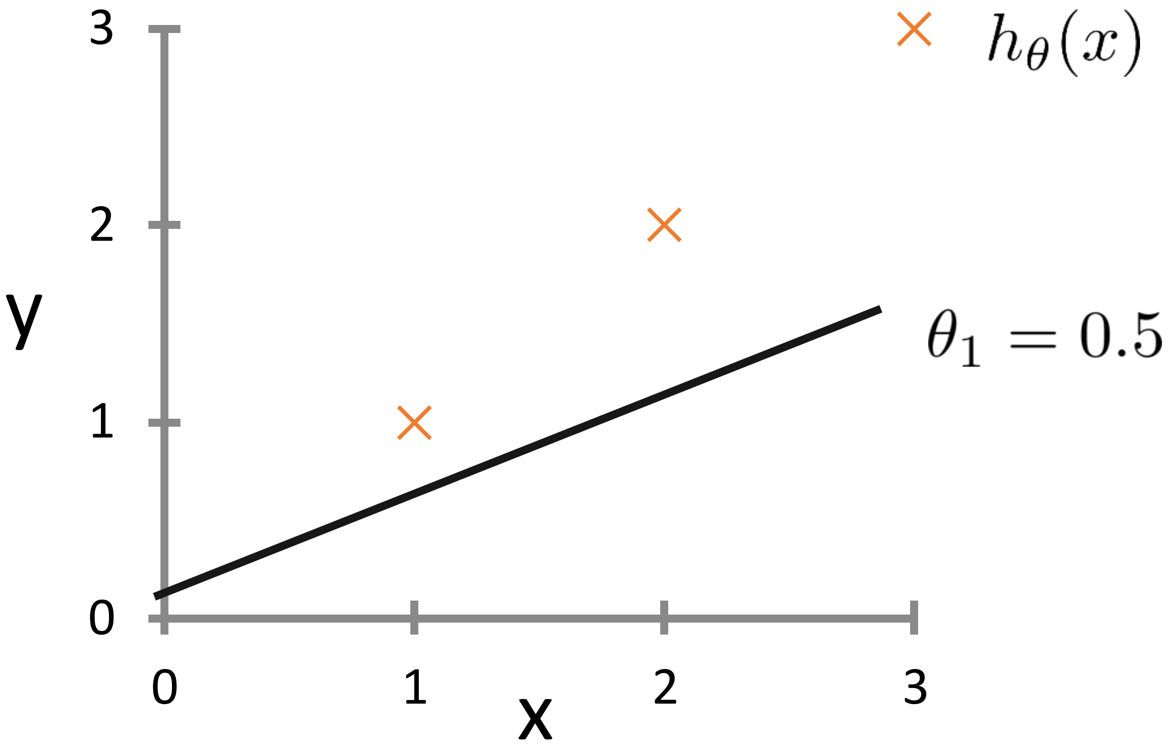




$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

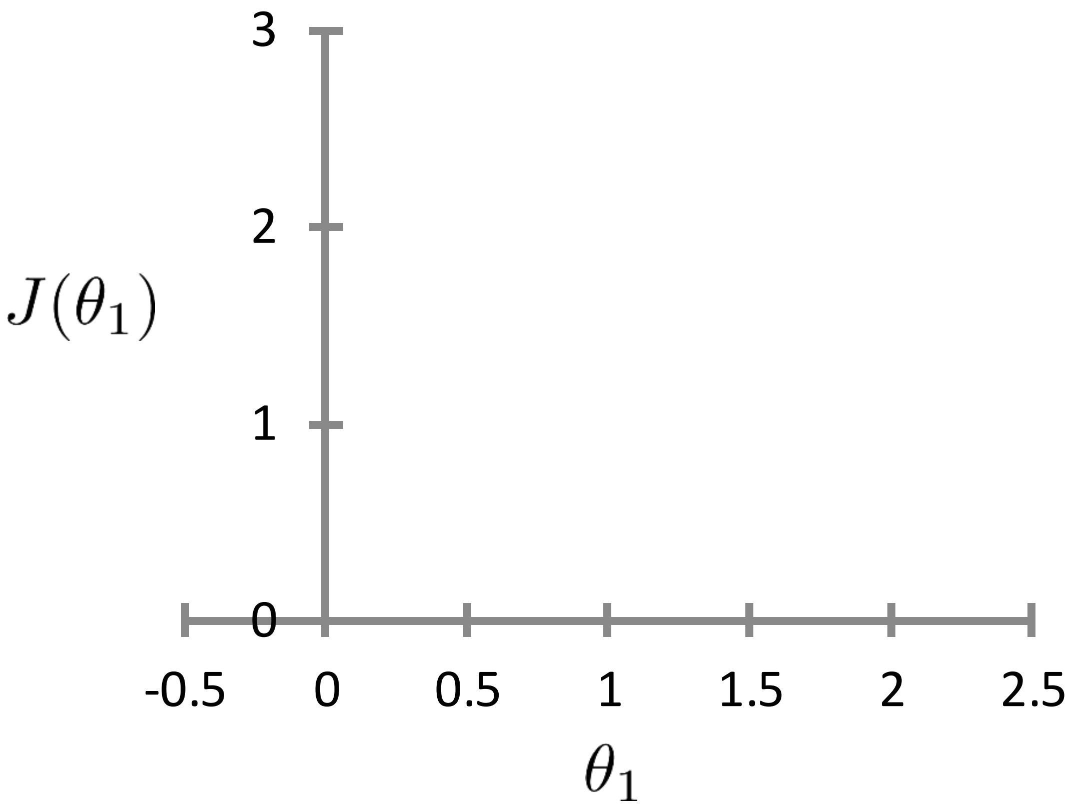
$h_\theta(x)$

(for fixed θ_1 , this is a function of x)



$J(\theta_1)$

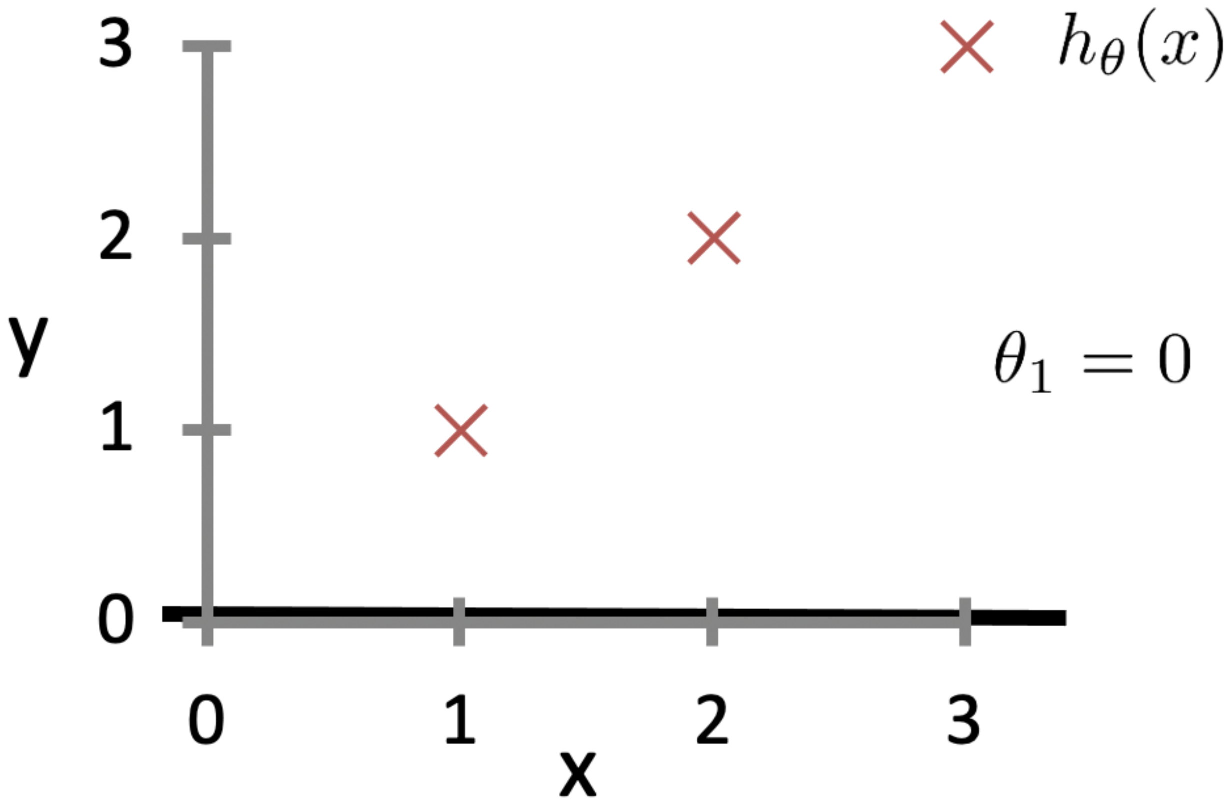
(function of the parameter θ_1)



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

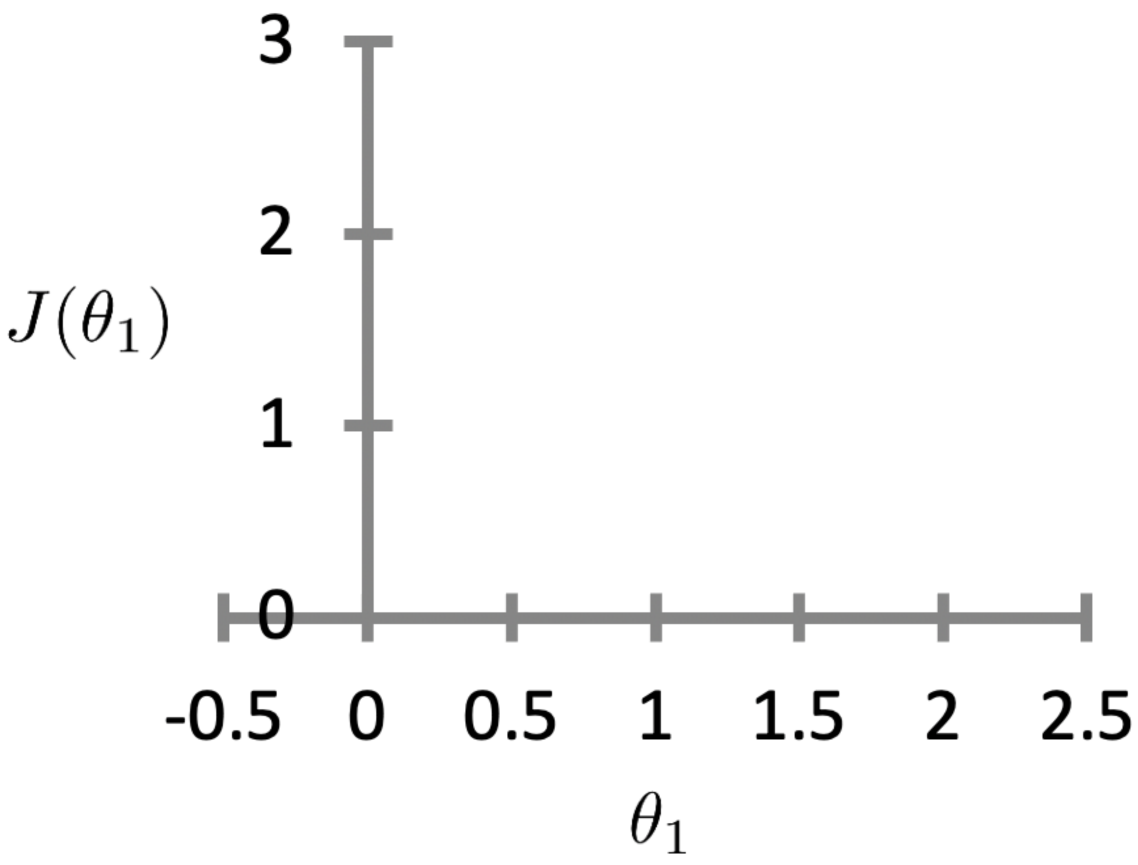
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



Linear Regression: Quick Review

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

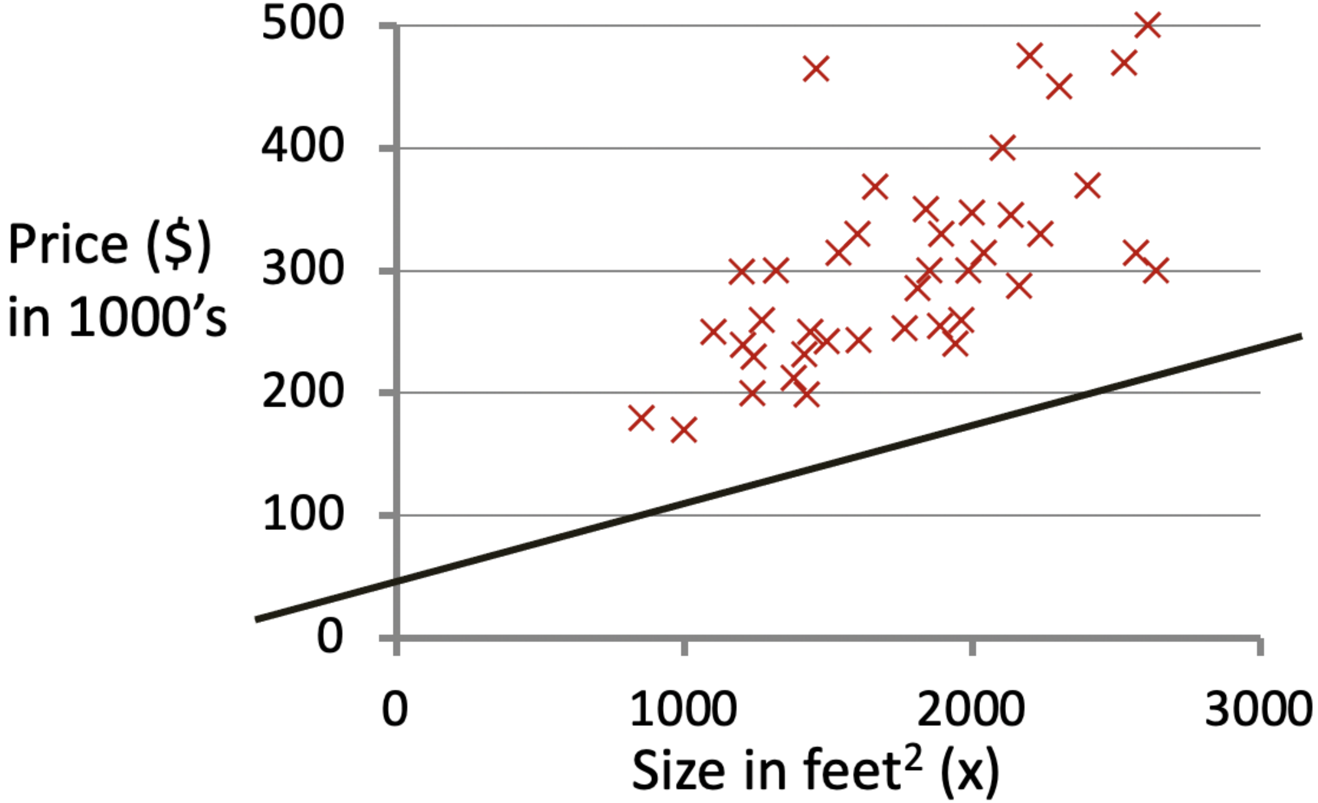
Parameters: θ_0, θ_1

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

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

$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$$h_\theta(x) = 50 + 0.06x$$

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

AI VIET NAM
@aivietnam.edu.vn

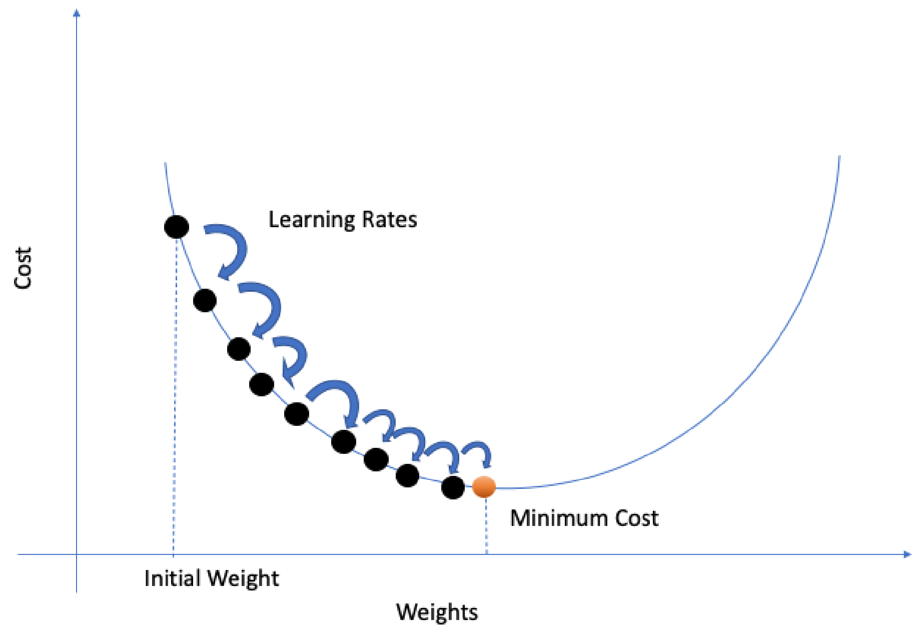
Linear Regression: Quick Review

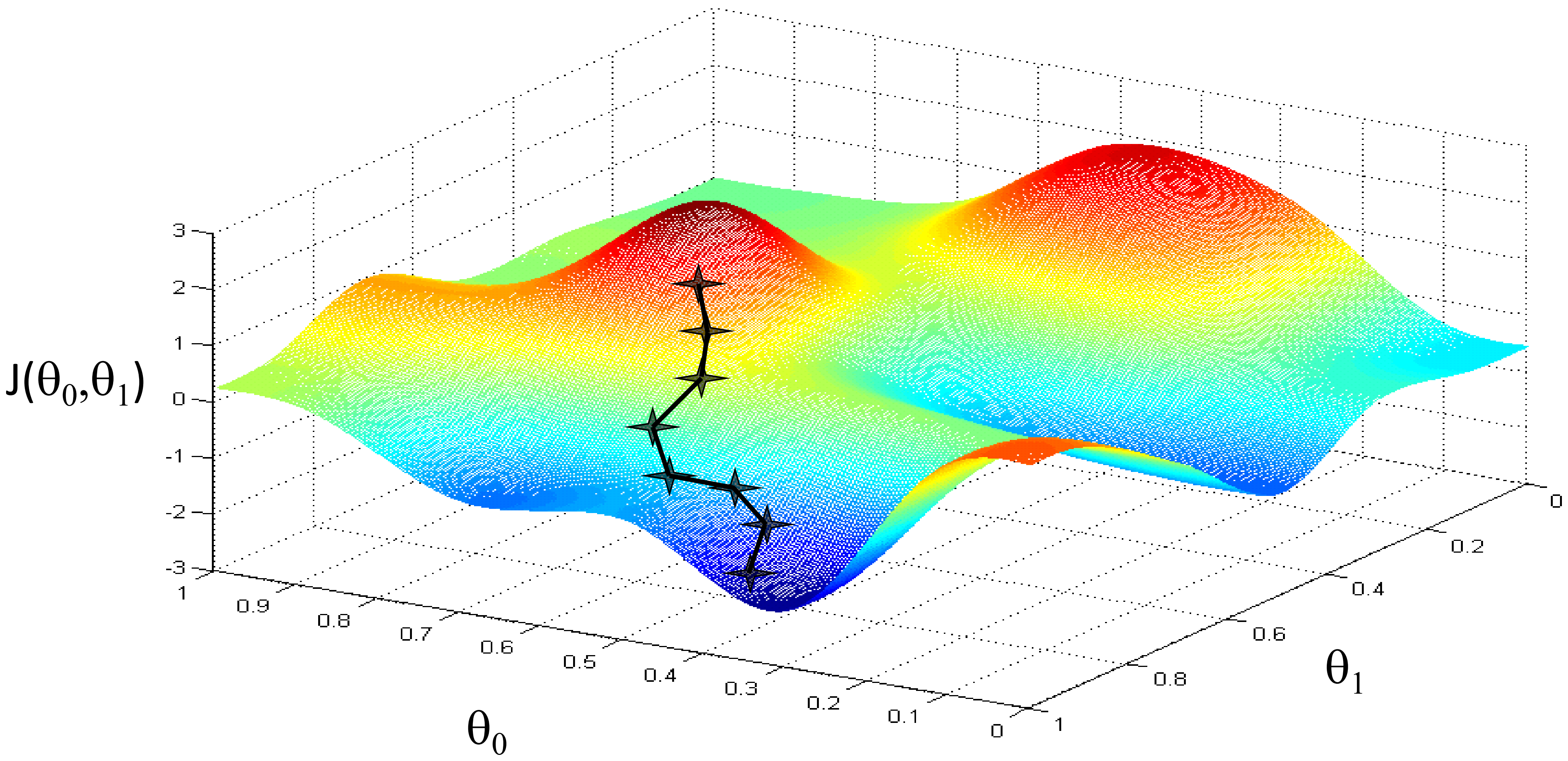
Have some function $J(\theta_0, \theta_1)$

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

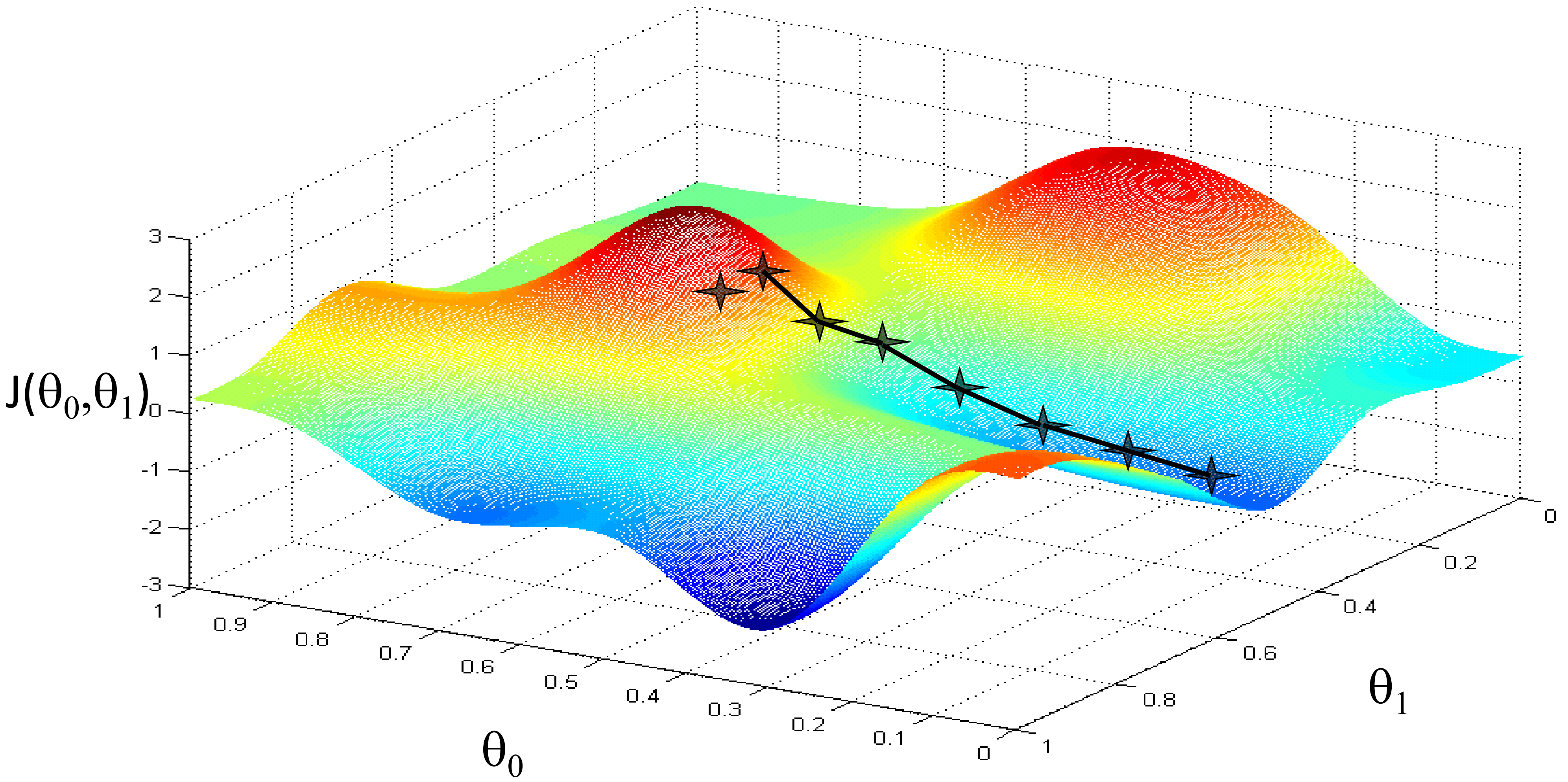
Outline:

- 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





3
2
1
0
-1
-2
-3
$J(\theta_0,\theta_1)$
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1



$J(\theta_0,\theta_1)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1

Linear Regression: Quick Review

Gradient descent algorithm

repeat until convergence {

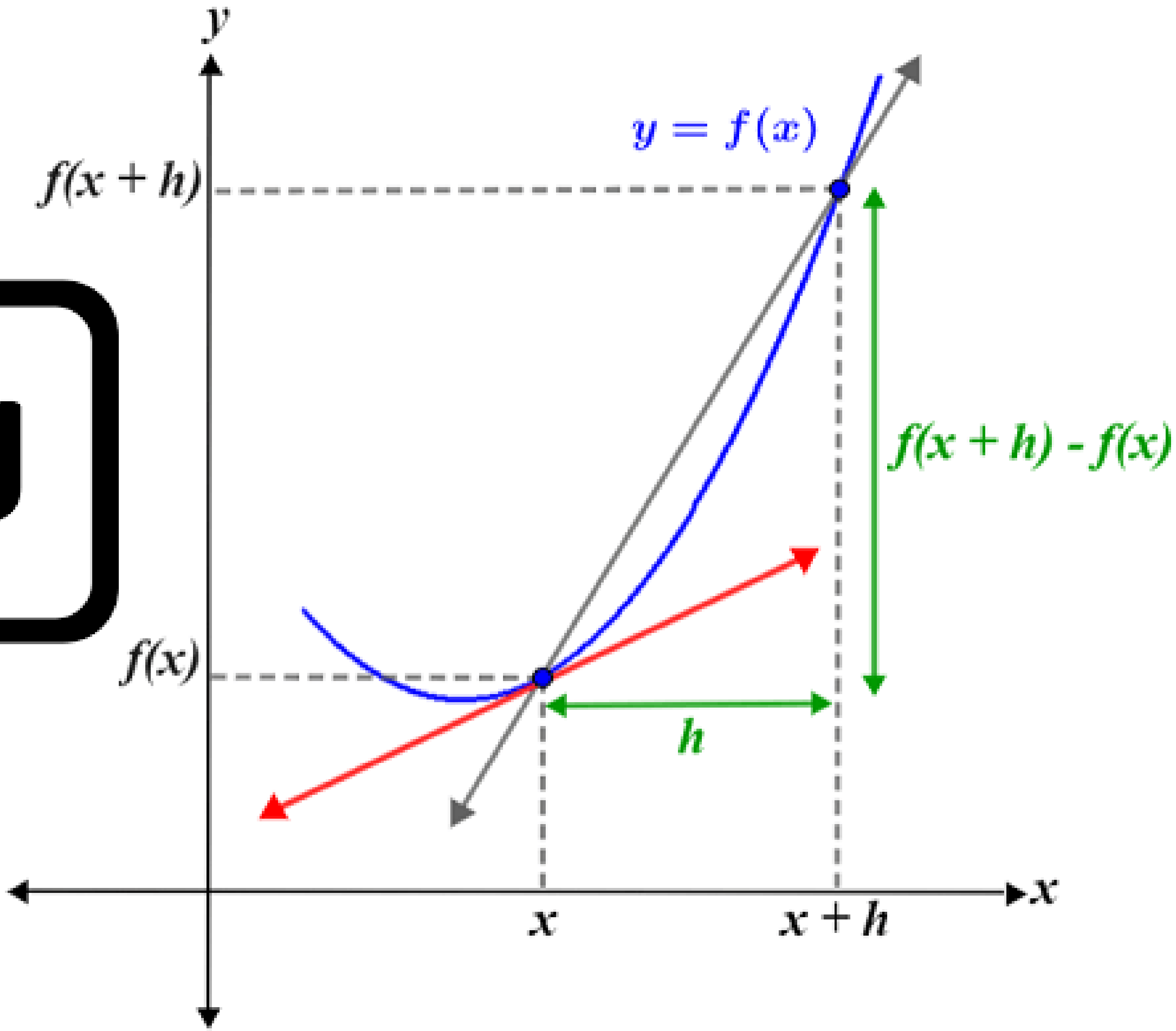
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

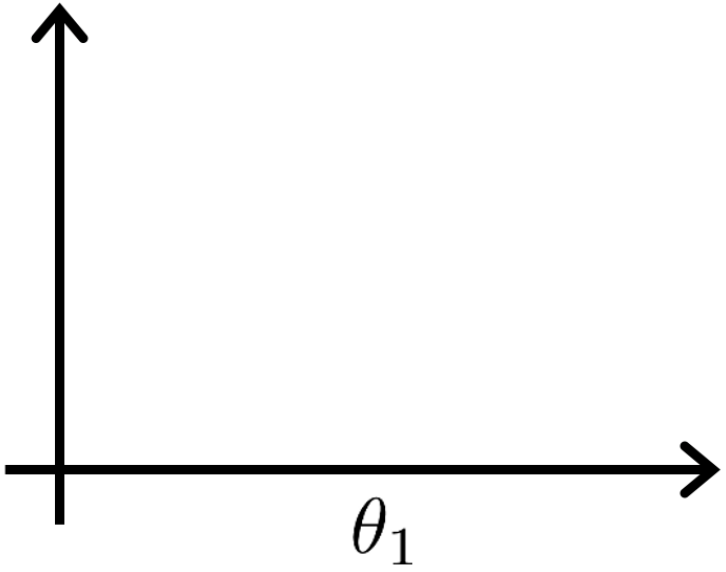
What, Why using a Derivative?



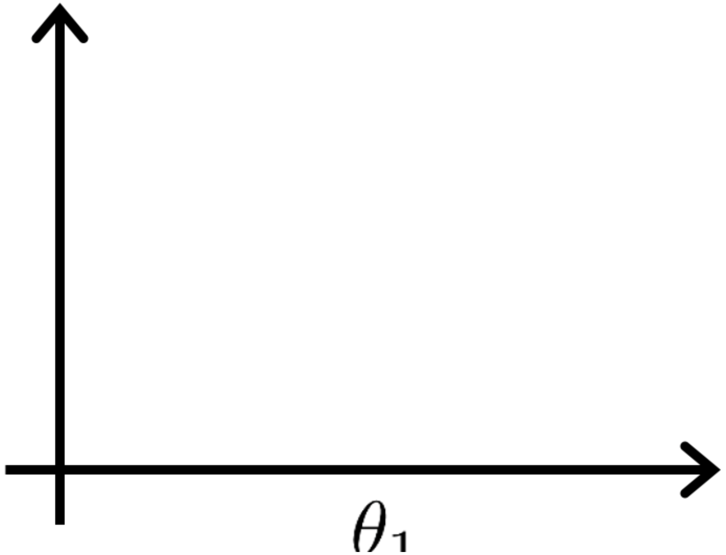
Linear Regression: Quick Review

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

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

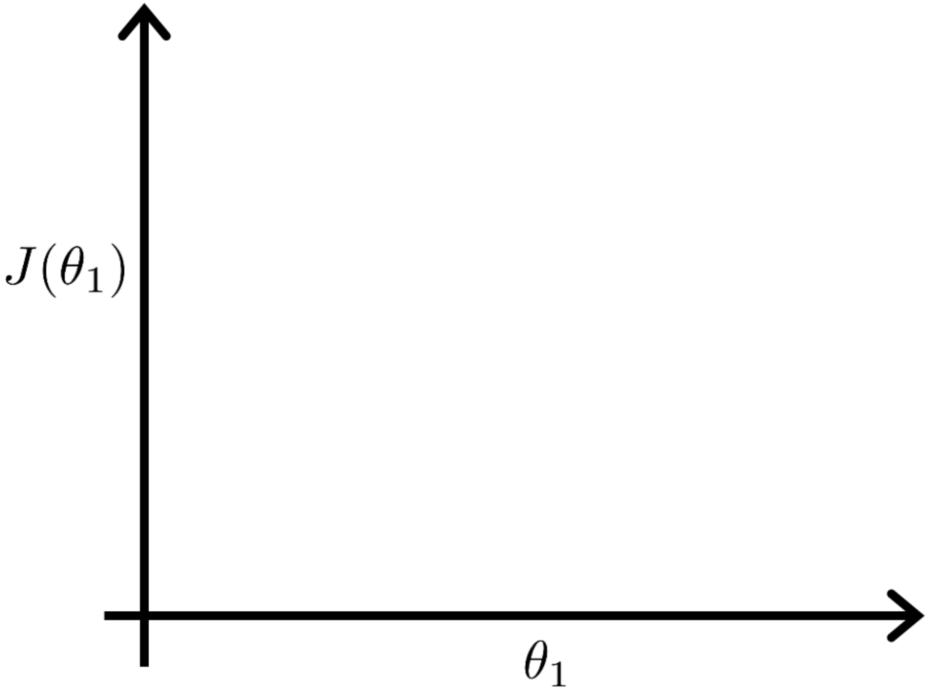


Linear Regression: Quick Review

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

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

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



Linear Regression: Quick Review

Gradient descent algorithm

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\
&\qquad (\text{for } j = 1 \text{ and } j = 0) \\
&\}
\end{aligned}
$$

Linear Regression Model

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

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

Linear Regression: Quick Review

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

$$\frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

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

AI VIET NAM
@aivietnam.edu.vn

Linear Regression: Quick Review

Gradient descent algorithm

$$\text{repeat until convergence } \{$$

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

$$\}$$

update θ_0 and θ_1 simultaneously

Linear Regression: Quick Review

“Batch” Gradient Descent

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

Linear Regression

Introduction

Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

House price data

Features / Label

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Advertising data

if area=6.0, price=?

if TV=55.0, Radio=34.0, and Newspaper=62.0, price=?

Features / Label

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9

Boston House Price Data

AI VIET NAM
@aivietnam.edu.vn

Linear Regression

❖ Area-based house price prediction

$$\text{predicted\_price} = \text{w} * \text{area} + b$$

$$\text{error} = (\text{predicted\_price} - \text{real\_price})^2$$

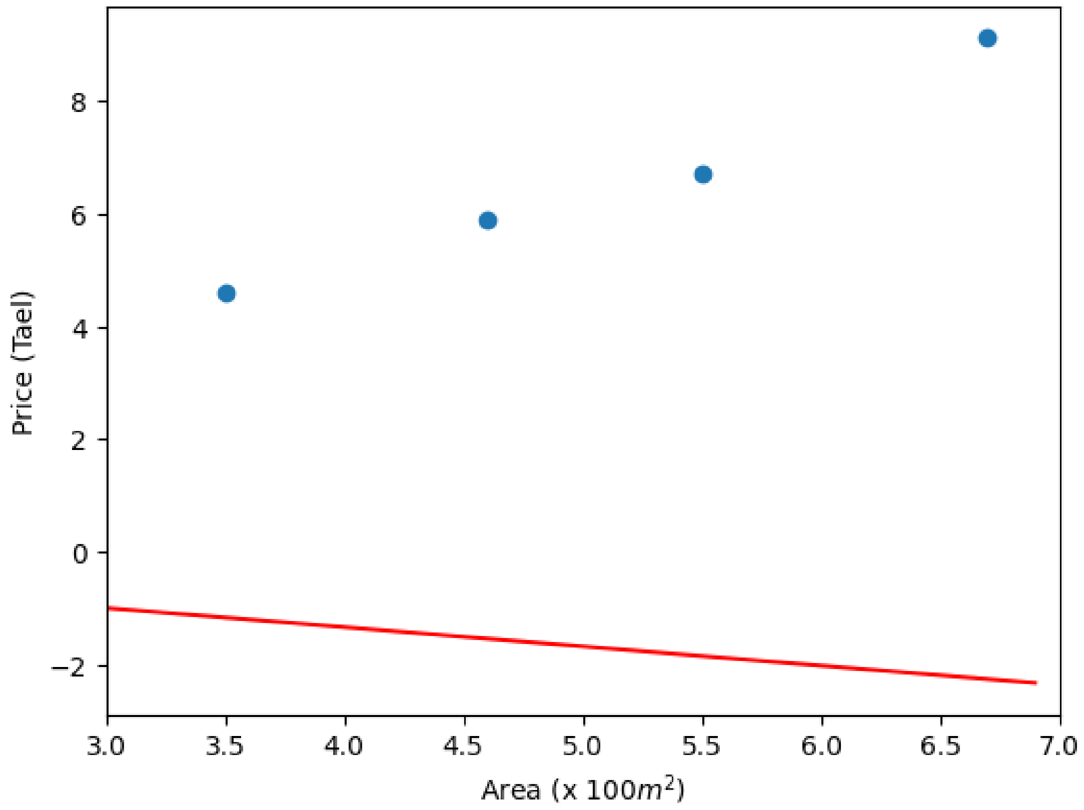
$$\hat{y}_i = wx_i + b$$

$$L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$$

area	price	predicted	error
6.7	9.1	-2.238	128.55
4.6	5.9	-1.524	55.11
3.5	4.6	-1.15	33.06
5.5	6.7	-1.83	72.76

w = -0.34

b = 0.04



AI VIET NAM
@aivietnam.edu.vn

Linear Regression

❖ Area-based house price prediction

$$\text{predicted\_price} = w * \text{area} + b$$

$$\text{error} = (\text{predicted\_price} - \text{real\_price})^2$$

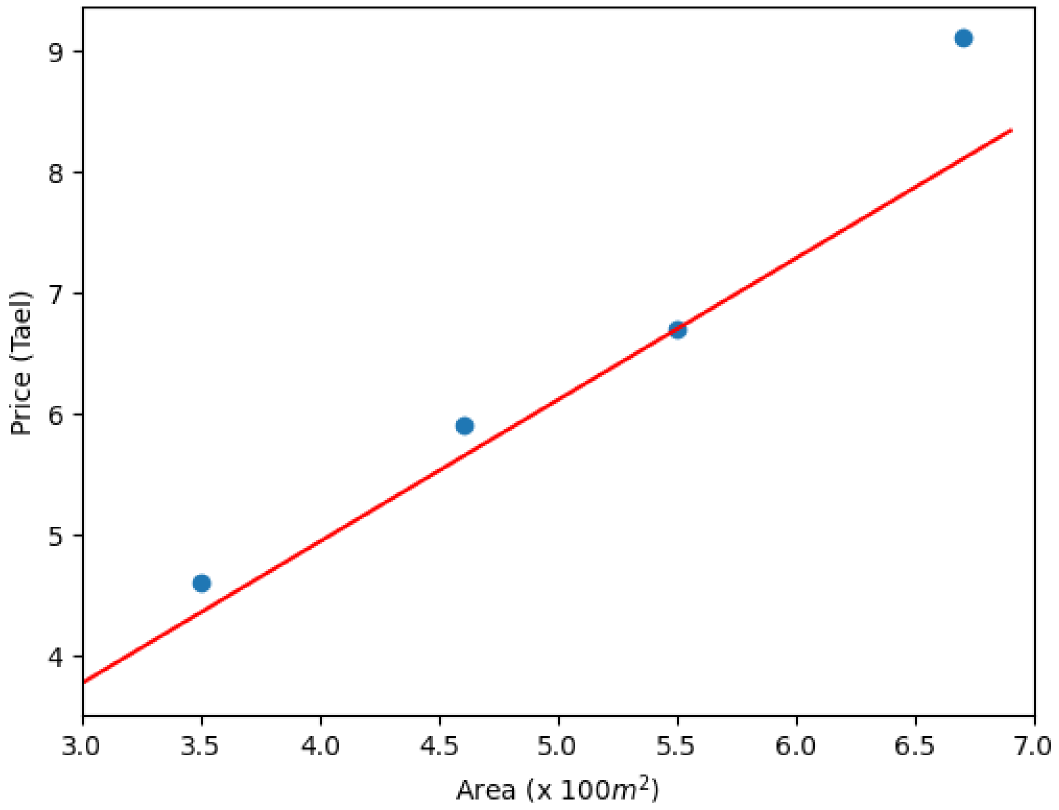
$$\hat{y}_i = wx_i + b$$

$$L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$$

area	price	predicted	error
6.7	9.1	8.099	1.002
4.6	5.9	5.642	0.066
3.5	4.6	4.355	0.06
5.5	6.7	6.695	0.00002

w = 1.17

b = 0.26



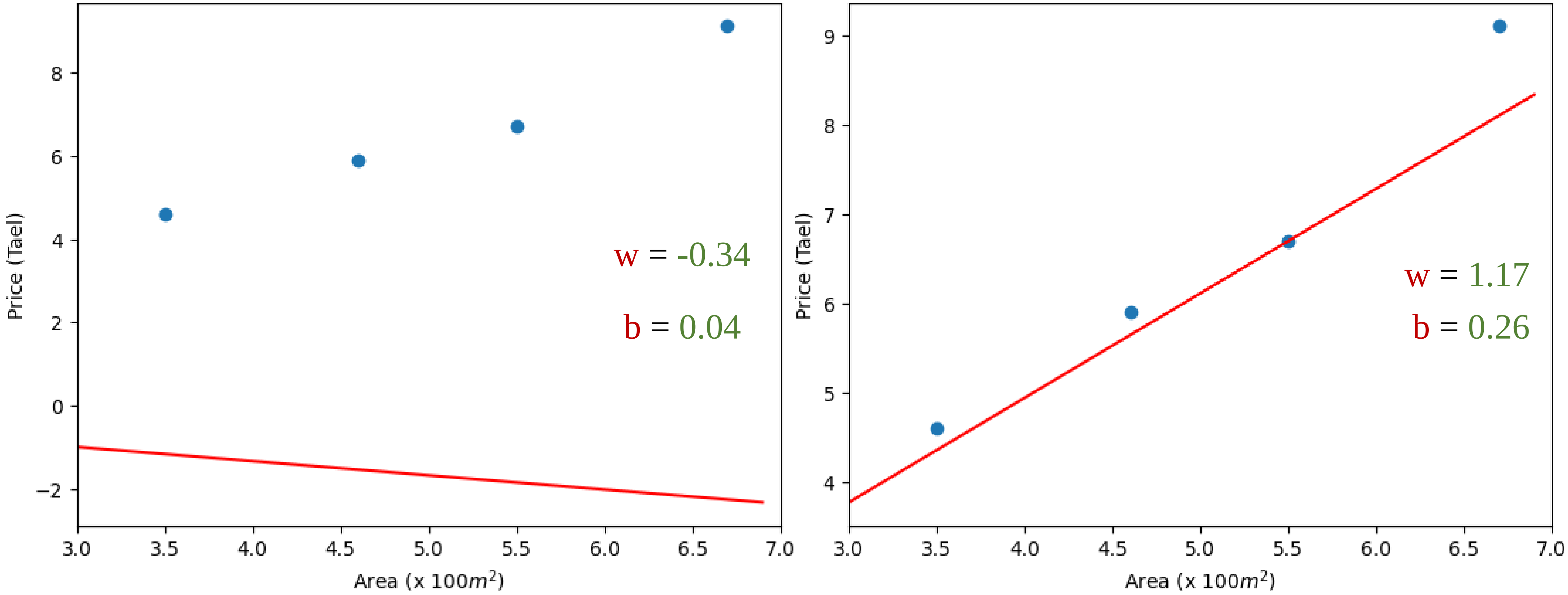
AI VIET NAM
@aivietnam.edu.vn

Linear Regression

❖ **Area-based house price prediction**

$$\hat{y}_i = wx_i + b$$
$$L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$$

How to change w and b so that $L(\hat{y}_i, y_i)$ reduces



AI VIET NAM
@aivietnam.edu.vn

Linear Regression

Linear equation

$$\hat{y} = wx + b$$

where $\hat{y}$ is a predicted value,

w and b are parameters

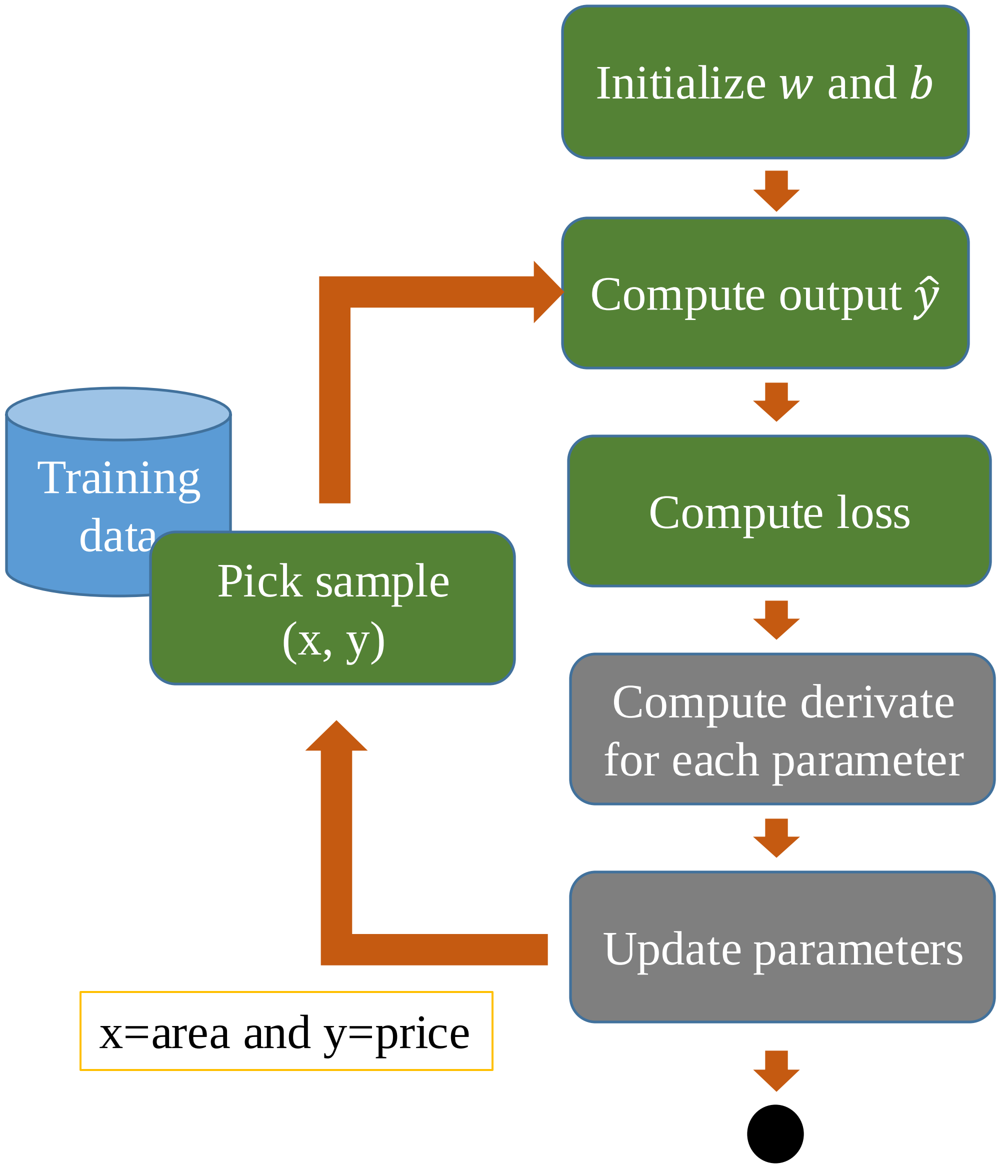
and x is input feature

Error (loss) computation

Idea: compare predicted values $\hat{y}$ and label values y

Squared loss

$$L(\hat{y}, y) = (\hat{y} - y)^2$$



How to find optimal w and b?

Linear Regression

Linear equation

$$\hat{y} = wx + b$$

where $\hat{y}$ is a predicted value,

w and b are parameters

and x is input feature

Error (loss) computation

Idea: compare predicted values $\hat{y}$ and label values y

Squared loss

$$\mathrm{L}(\hat{y}, y) = (\hat{y} - y)^2$$

How to find optimal w and b?

Use gradient descent to minimize the loss function

Compute derivate for each parameter

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w} = 2x(\hat{y} - y)$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b} = 2(\hat{y} - y)$$

Update parameters

$$w = w - \eta L'_w$$

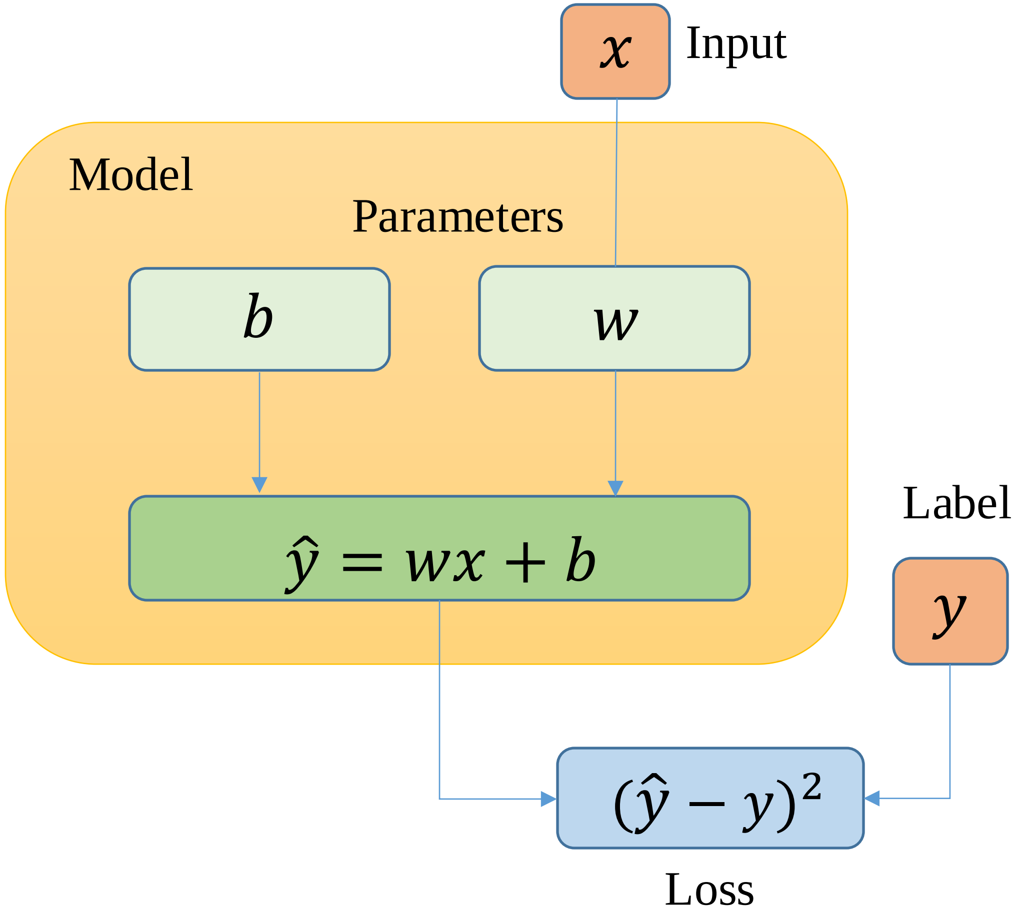
$$b = b - \eta L'_b$$

η is learning rate

Linear Regression

❖ Toy example

Diagram



Cheat sheet

Compute the output $\hat{y}$

$$\hat{y} = wx + b$$

Compute the loss

$$L = (\hat{y} - y)^2$$

Compute derivative

$$\frac{\partial L}{\partial w} = 2x(\hat{y} - y)$$

$$\frac{\partial L}{\partial b} = 2(\hat{y} - y)$$

Update parameters

$$w = w - \eta \frac{\partial L}{\partial w}$$

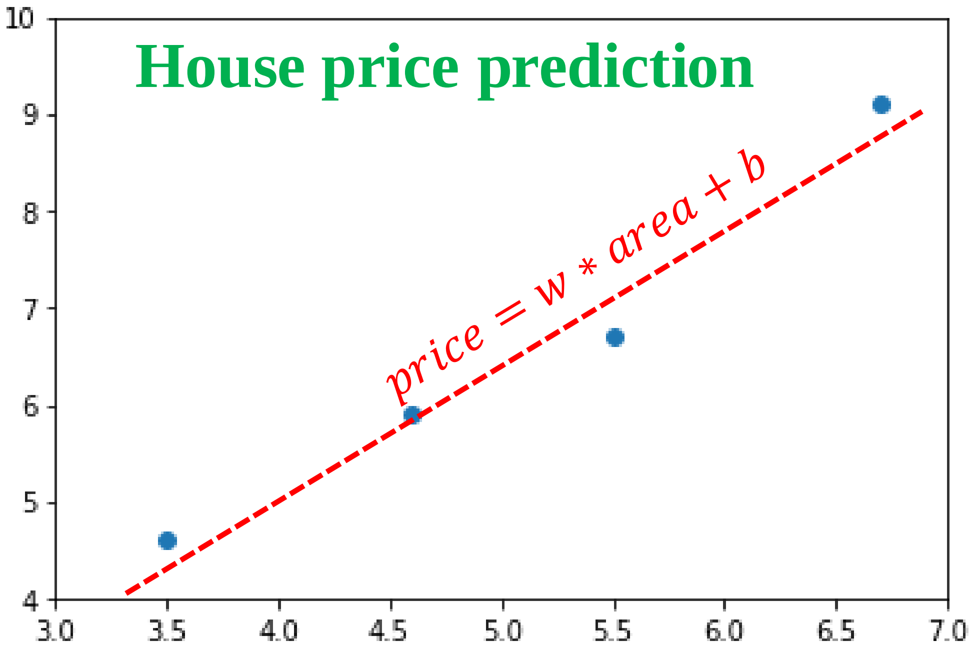
$$b = b - \eta \frac{\partial L}{\partial b}$$

AI VIET NAM
@aivietnam.edu.vn

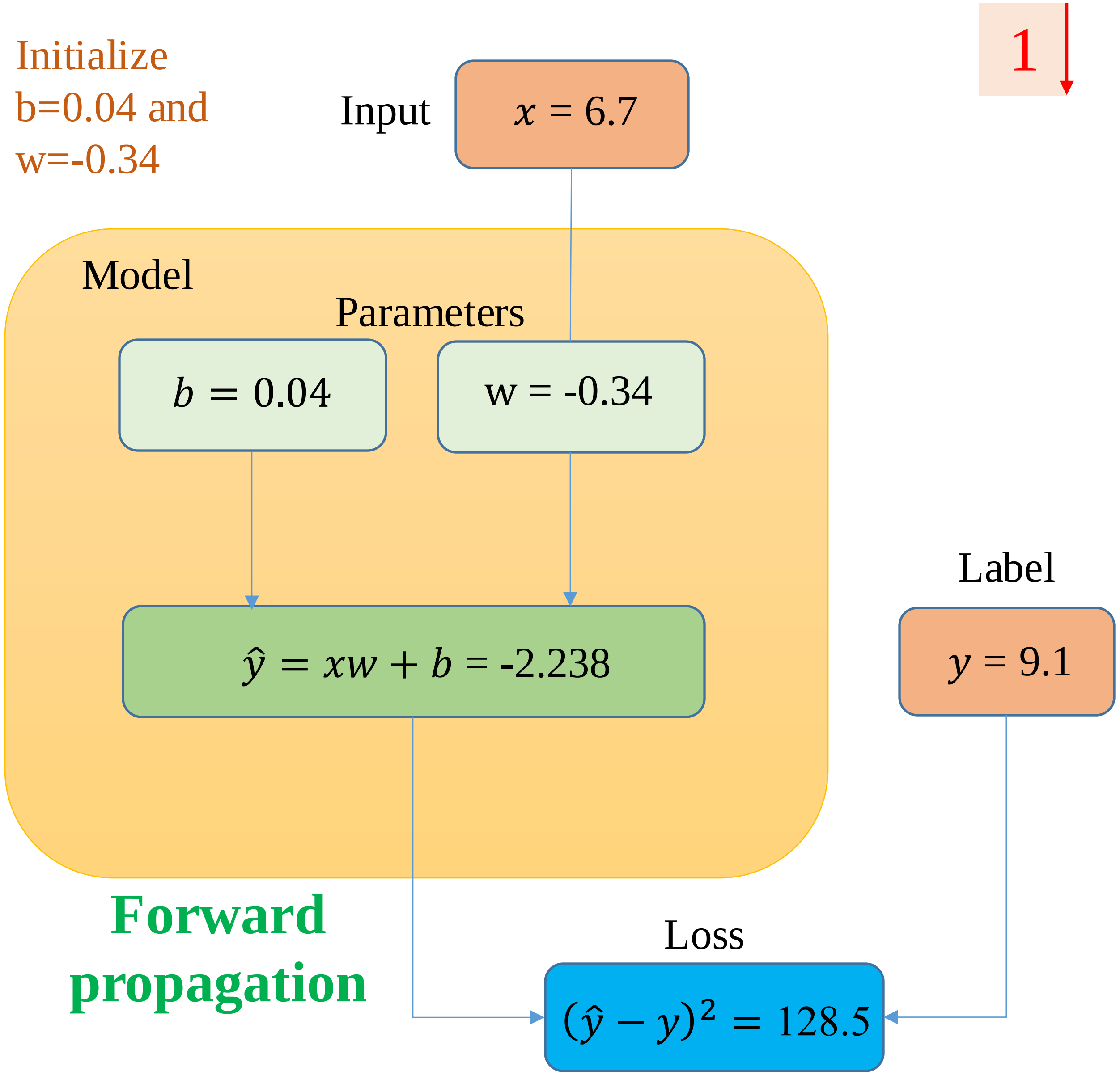
Linear Regression

Given sample data

Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

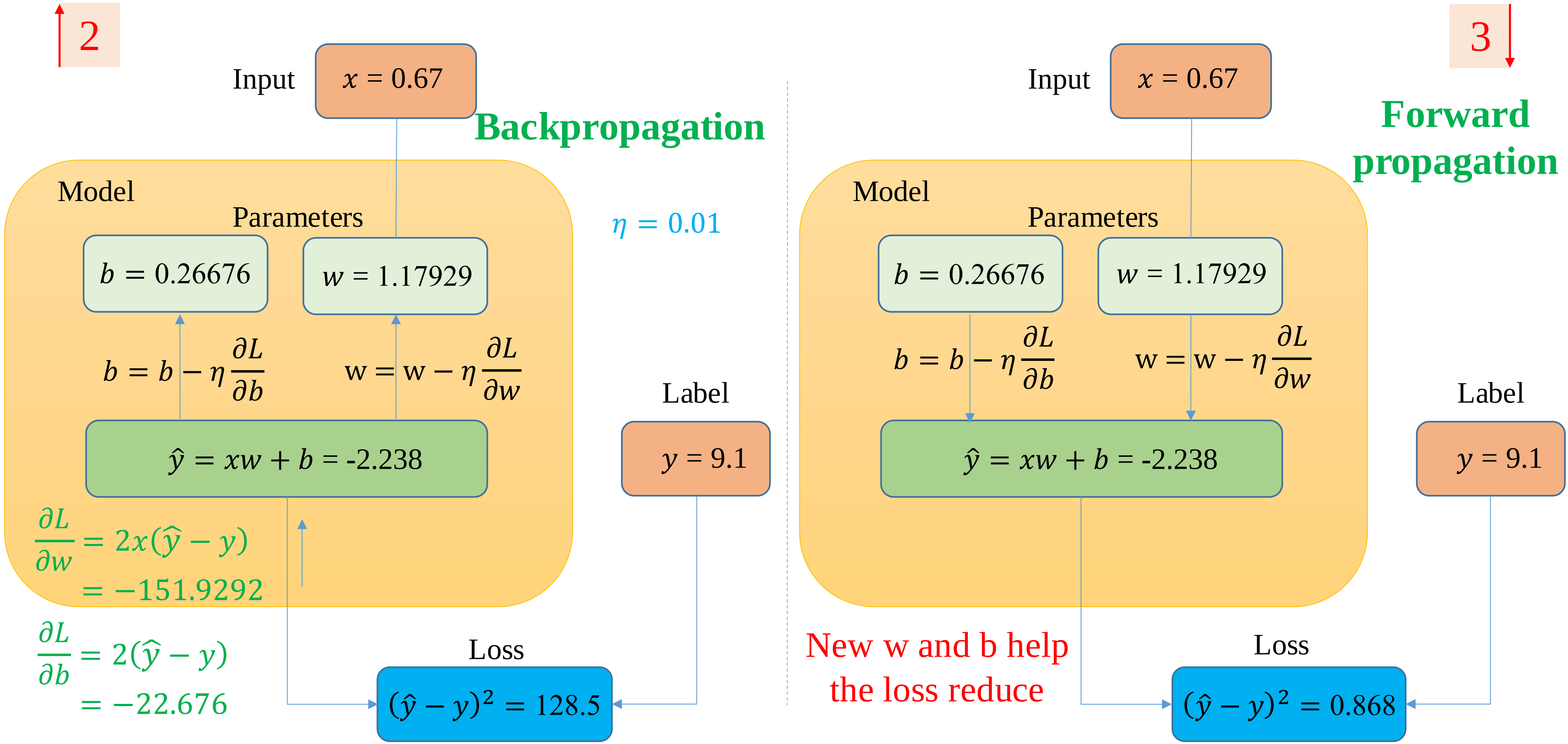


Initialize b=0.04 and w=-0.34



Forward propagation

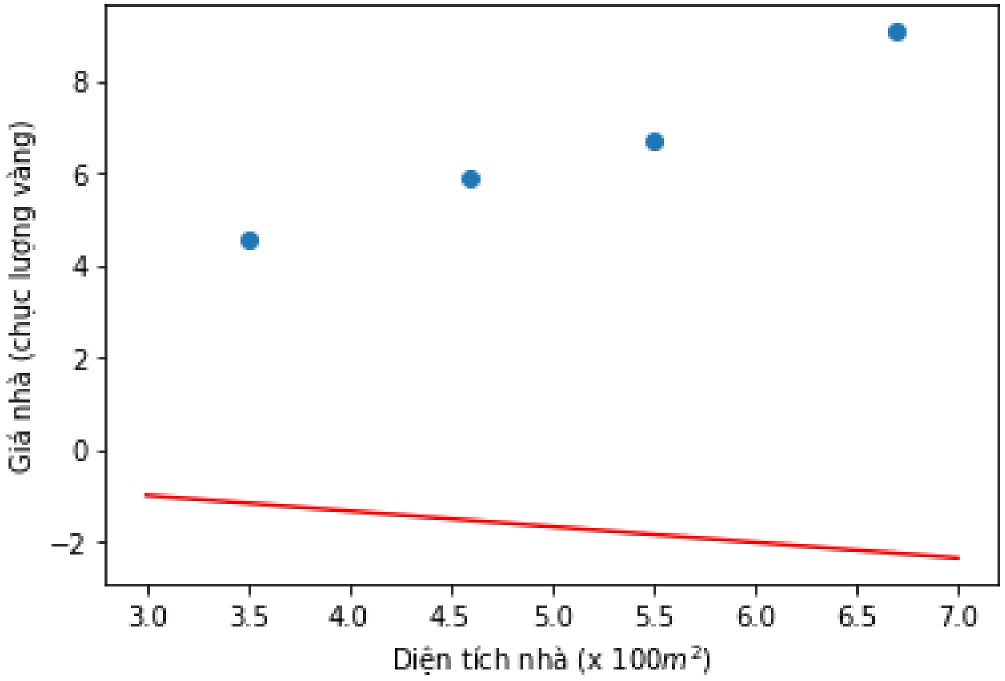
Linear Regression

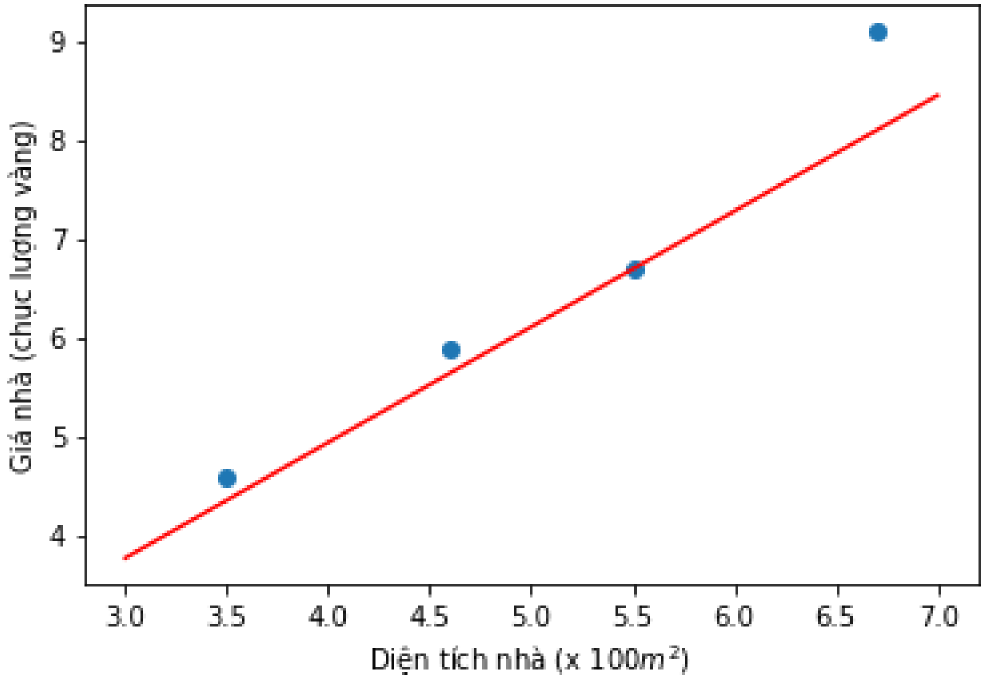


Linear Regression

❖ Toy example

Model prediction before and after the first update





w = -0.34 b = 0.04 L = 128.55

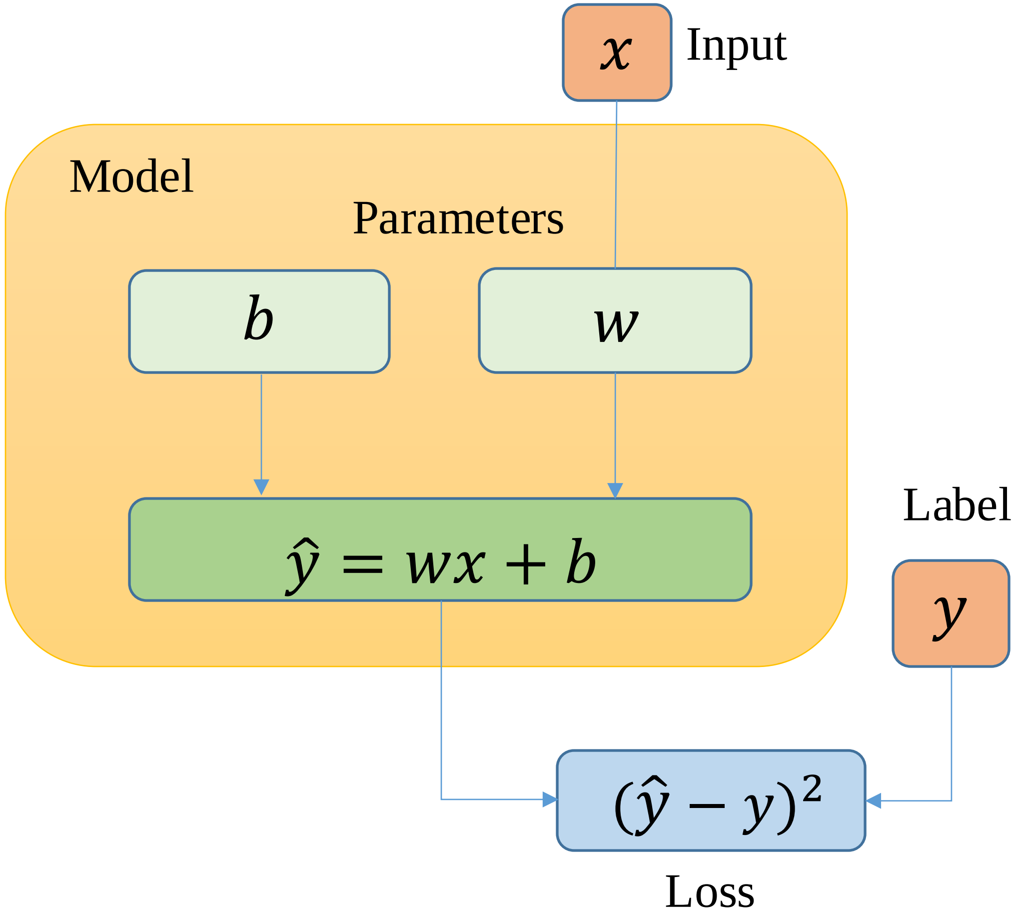
w = 1.179292 b = 0.26676 L = 0.868

Before updating

After updating

Linear Regression

❖ Summary (simple version)



1) Pick a sample (x, y) from training data

2) Compute the output $\hat{y}$

$$\hat{y} = wx + b$$

3) Compute loss

$$L = (\hat{y} - y)^2$$

4) Compute derivative

$$\frac{\partial L}{\partial w} = 2x(\hat{y} - y) \qquad \frac{\partial L}{\partial b} = 2(\hat{y} - y)$$

5) Update parameters

$$w = w - \eta \frac{\partial L}{\partial w} \qquad b = b - \eta \frac{\partial L}{\partial b}$$

η is learning rate

AI VIET NAM
@aivietnam.edu.vn

Linear Regression

❖ For the toy example

Cheat sheet

Compute the output $\hat{y}$

$$\hat{y} = wx + b$$

Compute the loss

$$L = (\hat{y} - y)^2$$

Compute derivative

$$\frac{\partial L}{\partial w} = 2x(\hat{y} - y)$$

$$\frac{\partial L}{\partial b} = 2(\hat{y} - y)$$

Update parameters

$$w = w - \eta \frac{\partial L}{\partial w}$$

$$b = b - \eta \frac{\partial L}{\partial b}$$

```python
# forward
def predict(x, w, b):
    return x*w + b

# compute gradient
def gradient(y_hat, y, x):
    dw = 2*x*(y_hat-y)
    db = 2*(y_hat-y)

    return (dw, db)

# update weights
def update_weight(w, b, lr, dw, db):
    w_new = w - lr*dw
    b_new = b - lr*db

    return (w_new, b_new)
```

Linear Regression

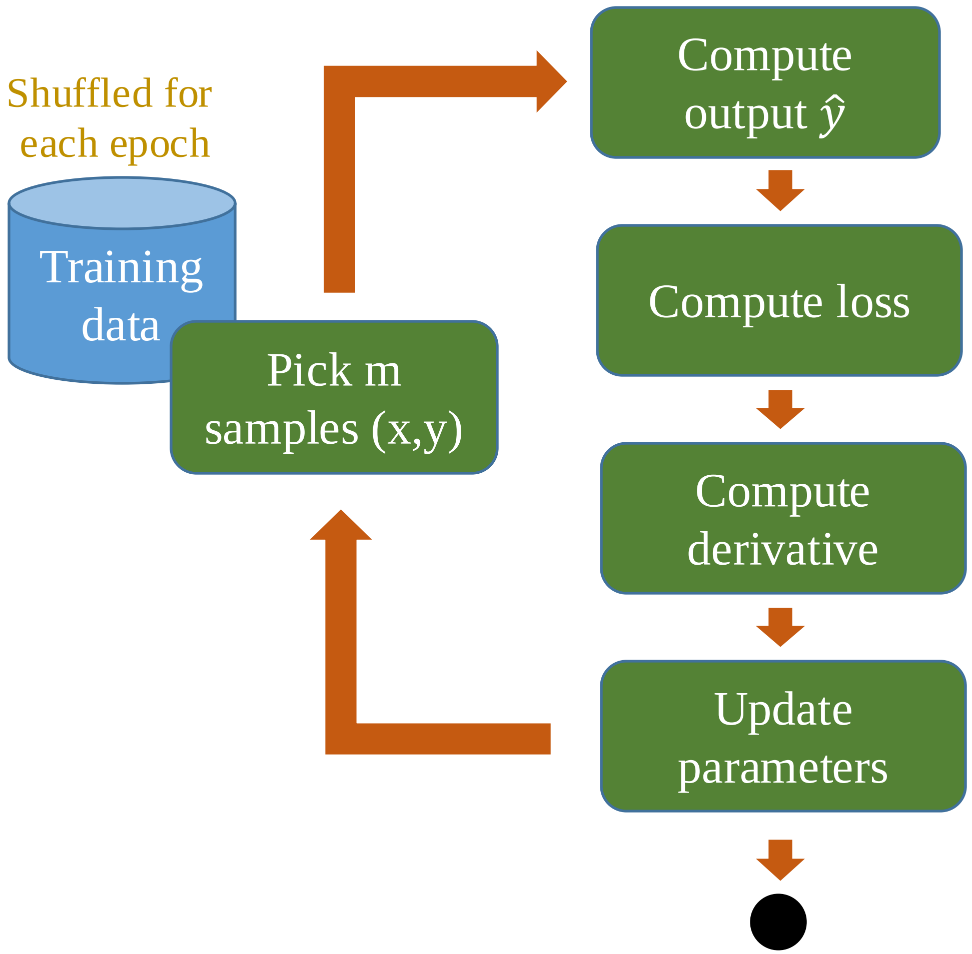
❖ Code for one update

```python
# forward
def predict(x, w, b):
    return x*w + b

# compute gradient
def gradient(y_hat, y, x):
    dw = 2*x*(y_hat-y)
    db = 2*(y_hat-y)

    return (dw, db)

# update weights
def update_weight(w, b, lr, dw, db):
    w_new = w - lr*dw
    b_new = b - lr*db

    return (w_new, b_new)
```

```python
# test sample
x = 6.7
y = 9.1

# init weights
b = 0.04
w = -0.34
lr = 0.01

# predict y_hat
y_hat = predict(x, w, b)
print('y_hat: ', y_hat)

# compute loss
loss = (y_hat-y)*(y_hat-y)
print('Loss: ', loss)

# compute gradient
(dw, db) = gradient(y_hat, y, x)
print('dw: ', dw)
print('db: ', db)

# update weights
(w, b) = update_weight(w, b, lr, dw, db)
print('w_new: ', w)
print('b_new: ', b)
```

Computational graph

- **House price prediction**
  - **m-sample training (1<m<N)**



1) Pick m samples $(x^{(i)}, y^{(i)})$ from training data

2) Tính output $\hat{y}^{(i)}$

$$\hat{y}^{(i)} = wx^{(i)} + b \qquad \text{for } 0 \leq i < m$$

3) Tính loss

$$L^{(i)} = (\hat{y}^{(i)} - y^{(i)})^2 \quad \text{for } 0 \leq i < m$$

4) Tính đạo hàm

$$\frac{\partial L^{(i)}}{\partial w} = 2x^{(i)}(\hat{y}^{(i)} - y^{(i)})$$

$$\frac{\partial L^{(i)}}{\partial b} = 2(\hat{y}^{(i)} - y^{(i)})$$

$$\text{for } 0 \leq i < m$$

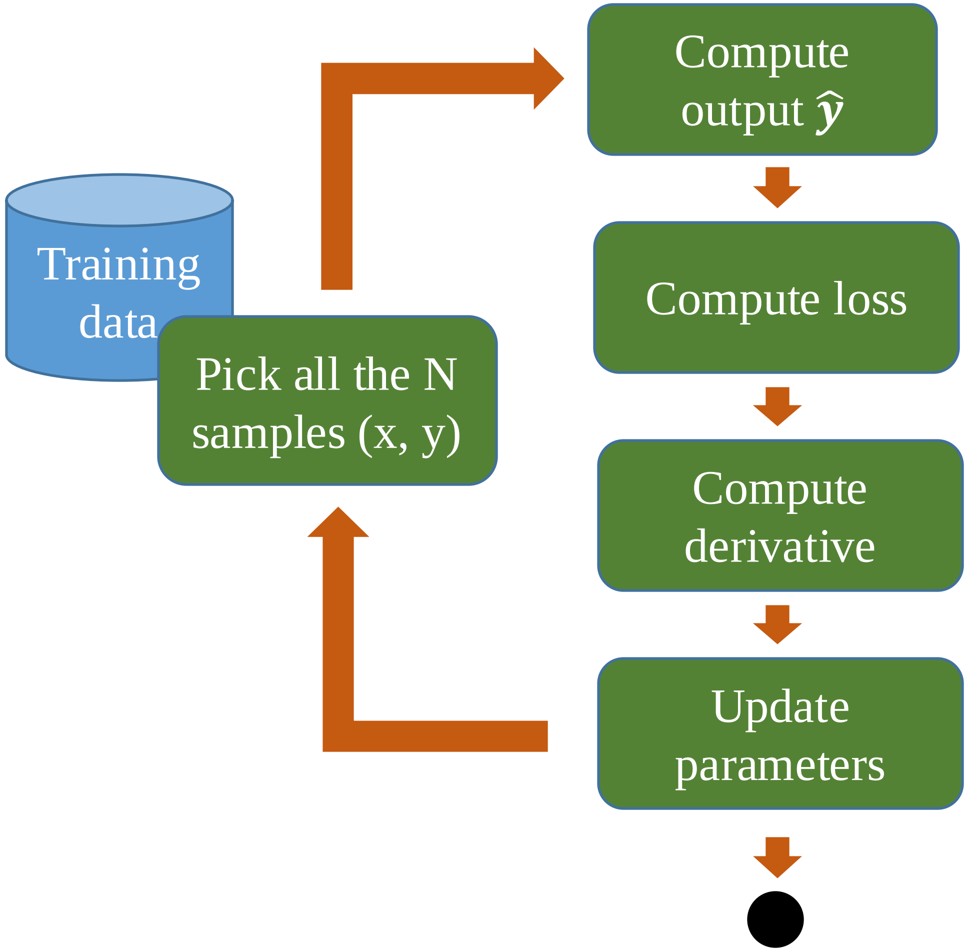
5) Cập nhật tham số

$$w = w - \eta \frac{\sum_i \frac{\partial L^{(i)}}{\partial w}}{m} \qquad b = b - \eta \frac{\sum_i \frac{\partial L^{(i)}}{\partial b}}{m}$$

Computational graph

❖ House price prediction

❖ N-sample training



1) Pick all the N samples $(x^{(i)}, y^{(i)})$ from training data

2) Tính output $\hat{y}^{(i)}$

$$\hat{y}^{(i)} = wx^{(i)} + b \quad \text{for } 0 \le i < N$$

3) Tính loss

$$L^{(i)} = (\hat{y}^{(i)} - y^{(i)})^2 \quad \text{for } 0 \le i < N$$

4) Tính đạo hàm

$$\frac{\partial L^{(i)}}{\partial w} = 2x^{(i)}(\hat{y}^{(i)} - y^{(i)})$$

$$\frac{\partial L^{(i)}}{\partial b} = 2(\hat{y}^{(i)} - y^{(i)}) \quad \text{for } 0 \le i < N$$

5) Cập nhật tham số

$$w = w - \eta \frac{\sum_i \frac{\partial L^{(i)}}{\partial w}}{N} \qquad b = b - \eta \frac{\sum_i \frac{\partial L^{(i)}}{\partial b}}{N}$$

Linear Regression

❖ Generalized formula

House price data

	Feature	Label
	area	price
	6.7	9.1
	4.6	5.9
	3.5	4.6
	5.5	6.7

Advertising data

Features			Label
TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Model

$$\text{price} = \text{w} * area + b$$

$$\hat{y} = wx + b$$

Model (vectorization)

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} \quad \text{where} \quad \boldsymbol{\theta}^T = [b \quad w]^T$$

$$\boldsymbol{x} = [x_0 \quad area]^T$$

$$x_0 = 1$$

Model

$$\text{Sale} = w_1 * TV + w_2 * Radio + w_3 * Newspaper + b$$

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Model (vectorization)

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} \quad \text{where} \quad \boldsymbol{\theta}^T = [b \quad w_1 \quad w_2 \quad w_3]^T$$

$$\boldsymbol{x} = [x_0 \quad TV \quad Radio \quad Newspaper]^T$$

$$x_0 = 1$$

Linear Regression

1) Pick a sample (x, y) from training data

2) Compute the output $\hat{y}$

$$\hat{y} = w_1 * TV + w_2 * R + w_3 * N + b$$

$$\hat{y} = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + b$$

3) Compute loss

$$L = (\hat{y} - y)^2$$

4) Compute derivative

$$\frac{\partial L}{\partial w_1} = 2x_1(\hat{y} - y) \qquad \frac{\partial L}{\partial w_3} = 2x_3(\hat{y} - y)$$

$$\frac{\partial L}{\partial w_2} = 2x_2(\hat{y} - y) \qquad \frac{\partial L}{\partial b} = 2(\hat{y} - y)$$

5) Update parameters

$$w_1 = w_1 - \eta \frac{\partial L}{\partial w_1} \qquad w_3 = w_3 - \eta \frac{\partial L}{\partial w_3}$$

$$w_2 = w_2 - \eta \frac{\partial L}{\partial w_2} \qquad b = b - \eta \frac{\partial L}{\partial b}$$

Features			Label
TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Advertising data

Model

$$\text{Sale} = w_1 * TV + w_2 * Radio + w_3 * Newspaper + b$$

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

AI VIET NAM
@aivietnam.edu.vn

Linear Regression

Features | Label

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

```
def initialize_params():
    w1 = random.gauss(mu=0.0, sigma=0.01)
    w2 = random.gauss(mu=0.0, sigma=0.01)
    w3 = random.gauss(mu=0.0, sigma=0.01)
    b  = 0

    return w1, w2, w3, b

# initialize model's parameters
w1, w2, w3, b = initialize_params()
```

```
# compute output and loss
def predict(x1, x2, x3, w1, w2, w3, b):
    return w1*x1 + w2*x2 + w3*x3 + b

def compute_loss(y_hat, y):
    return (y_hat - y)**2

# compute gradient
def compute_gradient_wi(xi, y, y_hat):
    dl_dwi = 2*xi*(y_hat-y)
    return dl_dwi

def compute_gradient_b(y, y_hat):
    dl_db = 2*(y_hat-y)
    return dl_db

# update weights
def update_weight_wi(wi, dl_dwi, lr):
    wi = wi - lr*dl_dwi
    return wi

def update_weight_b(b, dl_db, lr):
    b  = b - lr*dl_db
    return b
```

Outline

- Linear Regression Review
- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
- Other Discussions

Exercise 1

Bài tập 1 (kỹ thuật đọc và xử lý dữ liệu từ file .csv): Cho trước file dữ liệu advertising.csv, hãy hoàn thành function **prepare_data(file_name_dataset)** trả về dữ liệu đã được tổ chức (X cho input và y cho output).

```python

# dataset
import numpy as np
import matplotlib.pyplot as plt
import random

def get_column(data, index):

    #your code here ****************************

    return result

def prepare_data(file_name_dataset):
  data = np.genfromtxt(file_name_dataset, delimiter=',', skip_header=1).tolist()
  N = len(data)

  # get tv (index=0)
  tv_data = get_column(data, 0)

  # get radio (index=1)
  radio_data = get_column(data, 1)

```

Exercise 1

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9
8.7	48.9	75	7.2
57.5	32.8	23.5	11.8
120.2	19.6	11.6	13.2
8.6	2.1	1	4.8
199.8	2.6	21.2	15.6

X

Y

TV= X[0] = [230.1, 44.5, 17.2, 151.5, 180.8, 8.7, 57.5, 120.2, 8.6, 199.8]

Radio = X[1] = [37.8, 39.9, 45.9, 41.3, 10.8, 48.9, 32.8, 19.6, 2,1, 2.6]

News = X[2] = [69.2, 45.1, 69.3, 58.5, 58.4, 75, 32.5, 11.6, 1. 0, 21.2]

Sales = Y = [22.1, 10.4, 12, 16.5, 17.9, 7.2, 11.8, 13.2, 4.8, 15.6]

Exercise 1

```python
# dataset
import numpy as np
import matplotlib.pyplot as plt
import random

def get_column(data, index):
    result = [row[index] for row in data]
    return result

def prepare_data(file_name_dataset):
  data = np.genfromtxt(file_name_dataset, delimiter=',', skip_header=1).tolist()
  N = len(data)

  # get tv (index=0)
  tv_data = get_column(data, 0)

  # get radio (index=1)
  radio_data = get_column(data, 1)

  # get newspaper (index=2)
  newspaper_data = get_column(data, 2)

  # get sales (index=3)
  sales_data = get_column(data, 3)

  # building X input  and y output for training
  X = [tv_data, radio_data, newspaper_data]
  y = sales_data
  return X,y
```

Outline

- ➤ Linear Regression Review
- ➤ Exercise 1
- ➤ Exercise 2
- ➤ Exercise 3
- ➤ Exercise 4
- ➤ Exercise 5
- ➤ Other Discussions

Exercise 2

Bài tập 2 (kỹ thuật huấn luyện data dùng one sample - linear regression): Sử dụng kết quả dữ liệu đầu vào X, và dữ liệu đầu ra y từ bài 1, để phát triển chương trình dự đoán thông tin sales (y) từ X bằng cách dùng giải thuật linear regression with one sample-training với loss được tính bằng công thức Mean Squared Error L = $(\hat{y} - y)^2$. Sơ đồ hoạt động của giải thuật được mô tả ở hình 2. Nhiệm vụ của bạn là hoàn thành function **implement_linear_regression(X_data, y_data, epoch_max, lr)** và trả về 4 tham số w1,w2,w3,b và lịch sử tính loss như bên dưới.

1) Pick a sample (x, y) from training data

2) Tính output $\hat{y}$

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

3) Tính loss

$$L = (\hat{y} - y)^2$$

4) Tính đạo hàm

$$L'_{w_1} = 2x_1(\hat{y} - y)$$
$$L'_{w_2} = 2x_2(\hat{y} - y)$$
$$L'_{w_3} = 2x_3(\hat{y} - y)$$
$$L'_b = 2(\hat{y} - y)$$

5) Cập nhật tham số

η is learning rate

$$w_1 = w_1 - \eta L'_{w_1}$$
$$w_2 = w_2 - \eta L'_{w_2}$$
$$w_3 = w_3 - \eta L'_{w_3}$$
$$b = b - \eta L'_b$$

Hình 2: Các bước để thực hiện train linear regression model

AI VIET NAM
@aivietnam.edu.vn

Exercise 2

```python
def implement_linear_regression(X_data, y_data, epoch_max = 50,lr = 1e-5):
    losses = []

    w1, w2, w3, b = initialize_params()

    N = len(y_data)
    for epoch in range(epoch_max):
        for i in range(N):
            # get a sample
            x1 = X_data[0][i]
            x2 = X_data[1][i]
            x3 = X_data[2][i]

            y  = y_data[i]

            # print(y)
            # compute output
            y_hat = predict(x1, x2, x3, w1, w2, w3, b)

            # compute loss
            loss = compute_loss_mse(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dw1 = compute_gradient_wi(x1, y, y_hat)
            dl_dw2 = compute_gradient_wi(x2, y, y_hat)
            dl_dw3 = compute_gradient_wi(x3, y, y_hat)
            dl_db  = compute_gradient_b(y, y_hat)

            # update parameters
            w1 = update_weight_wi(w1, dl_dw1, lr)
            w2 = update_weight_wi(w2, dl_dw2, lr)
            w3 = update_weight_wi(w3, dl_dw3, lr)
            b  = update_weight_b(b, dl_db, lr)

            # logging
            losses.append(loss)
    return (w1,w2,w3,b, losses)
```

```python
def initialize_params():
    # w1 = random.gauss(mu=0.0, sigma=0.01)
    # w2 = random.gauss(mu=0.0, sigma=0.01)
    # w3 = random.gauss(mu=0.0, sigma=0.01)
    # b  = 0

    w1, w2, w3, b = (0.016992259082509283, 0.0070783670518262355,
        -0.002307860847821344, 0)
    return w1, w2, w3, b
```

Exercise 2

```python
def implement_linear_regression(X_data, y_data, epoch_max = 50,lr = 1e-5):
    losses = []

    w1, w2, w3, b = initialize_params()

    N = len(y_data)
    for epoch in range(epoch_max):
        for i in range(N):
            # get a sample
            x1 = X_data[0][i]
            x2 = X_data[1][i]
            x3 = X_data[2][i]

            y  = y_data[i]

            # print(y)
            # compute output
            y_hat = predict(x1, x2, x3, w1, w2, w3, b)

            # compute loss
            loss = compute_loss_mse(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dw1 = compute_gradient_wi(x1, y, y_hat)
            dl_dw2 = compute_gradient_wi(x2, y, y_hat)
            dl_dw3 = compute_gradient_wi(x3, y, y_hat)
            dl_db  = compute_gradient_b(y, y_hat)

            # update parameters
            w1 = update_weight_wi(w1, dl_dw1, lr)
            w2 = update_weight_wi(w2, dl_dw2, lr)
            w3 = update_weight_wi(w3, dl_dw3, lr)
            b  = update_weight_b(b, dl_db, lr)

            # logging
            losses.append(loss)
    return (w1,w2,w3,b, losses)
```

```python
# compute output and loss
def predict(x1, x2, x3, w1, w2, w3, b):
    return w1*x1 + w2*x2 + w3*x3 + b
```

AI VIET NAM
@aivietnam.edu.vn

Exercise 2

```python
def implement_linear_regression(X_data, y_data, epoch_max = 50,lr = 1e-5):
    losses = []

    w1, w2, w3, b = initialize_params()

    N = len(y_data)
    for epoch in range(epoch_max):
        for i in range(N):
            # get a sample
            x1 = X_data[0][i]
            x2 = X_data[1][i]
            x3 = X_data[2][i]

            y  = y_data[i]

            # print(y)
            # compute output
            y_hat = predict(x1, x2, x3, w1, w2, w3, b)

            # compute loss
            loss = compute_loss_mse(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dw1 = compute_gradient_wi(x1, y, y_hat)
            dl_dw2 = compute_gradient_wi(x2, y, y_hat)
            dl_dw3 = compute_gradient_wi(x3, y, y_hat)
            dl_db  = compute_gradient_b(y, y_hat)

            # update parameters
            w1 = update_weight_wi(w1, dl_dw1, lr)
            w2 = update_weight_wi(w2, dl_dw2, lr)
            w3 = update_weight_wi(w3, dl_dw3, lr)
            b  = update_weight_b(b, dl_db, lr)

            # logging
            losses.append(loss)
    return (w1,w2,w3,b, losses)
```

```python
def compute_loss_mse(y_hat, y):
    return (y_hat - y)**2
```

Exercise 2

```python
def implement_linear_regression(X_data, y_data, epoch_max = 50,lr = 1e-5):
    losses = []

    w1, w2, w3, b = initialize_params()

    N = len(y_data)
    for epoch in range(epoch_max):
        for i in range(N):
            # get a sample
            x1 = X_data[0][i]
            x2 = X_data[1][i]
            x3 = X_data[2][i]

            y  = y_data[i]

            # print(y)
            # compute output
            y_hat = predict(x1, x2, x3, w1, w2, w3, b)

            # compute loss
            loss = compute_loss_mse(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dw1 = compute_gradient_wi(x1, y, y_hat)
            dl_dw2 = compute_gradient_wi(x2, y, y_hat)
            dl_dw3 = compute_gradient_wi(x3, y, y_hat)
            dl_db  = compute_gradient_b(y, y_hat)

            # update parameters
            w1 = update_weight_wi(w1, dl_dw1, lr)
            w2 = update_weight_wi(w2, dl_dw2, lr)
            w3 = update_weight_wi(w3, dl_dw3, lr)
            b  = update_weight_b(b, dl_db, lr)

            # logging
            losses.append(loss)
    return (w1,w2,w3,b, losses)
```

```python
# compute gradient
def compute_gradient_wi(xi, y, y_hat):
    dl_dwi = 2*xi*(y_hat-y)
    return dl_dwi
```

Exercise 2

```python
def implement_linear_regression(X_data, y_data, epoch_max = 50,lr = 1e-5):
    losses = []

    w1, w2, w3, b = initialize_params()

    N = len(y_data)
    for epoch in range(epoch_max):
        for i in range(N):
            # get a sample
            x1 = X_data[0][i]
            x2 = X_data[1][i]
            x3 = X_data[2][i]

            y  = y_data[i]

            # print(y)
            # compute output
            y_hat = predict(x1, x2, x3, w1, w2, w3, b)

            # compute loss
            loss = compute_loss_mse(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dw1 = compute_gradient_wi(x1, y, y_hat)
            dl_dw2 = compute_gradient_wi(x2, y, y_hat)
            dl_dw3 = compute_gradient_wi(x3, y, y_hat)
            dl_db  = compute_gradient_b(y, y_hat)

            # update parameters
            w1 = update_weight_wi(w1, dl_dw1, lr)
            w2 = update_weight_wi(w2, dl_dw2, lr)
            w3 = update_weight_wi(w3, dl_dw3, lr)
            b  = update_weight_b(b, dl_db, lr)

            # logging
            losses.append(loss)
    return (w1,w2,w3,b, losses)
```

```python
def compute_gradient_b(y, y_hat):
    dl_db = 2*(y_hat-y)
    return dl_db
```

Exercise 2

```python
def implement_linear_regression(X_data, y_data, epoch_max = 50,lr = 1e-5):
  losses = []

  w1, w2, w3, b = initialize_params()

  N = len(y_data)
  for epoch in range(epoch_max):
      for i in range(N):
          # get a sample
          x1 = X_data[0][i]
          x2 = X_data[1][i]
          x3 = X_data[2][i]

          y  = y_data[i]

          # print(y)
          # compute output
          y_hat = predict(x1, x2, x3, w1, w2, w3, b)

          # compute loss
          loss = compute_loss_mse(y, y_hat)

          # compute gradient w1, w2, w3, b
          dl_dw1 = compute_gradient_wi(x1, y, y_hat)
          dl_dw2 = compute_gradient_wi(x2, y, y_hat)
          dl_dw3 = compute_gradient_wi(x3, y, y_hat)
          dl_db  = compute_gradient_b(y, y_hat)

          # update parameters
          w1 = update_weight_wi(w1, dl_dw1, lr)
          w2 = update_weight_wi(w2, dl_dw2, lr)
          w3 = update_weight_wi(w3, dl_dw3, lr)
          b  = update_weight_b(b, dl_db, lr)

          # logging
          losses.append(loss)
  return (w1,w2,w3,b, losses)
```

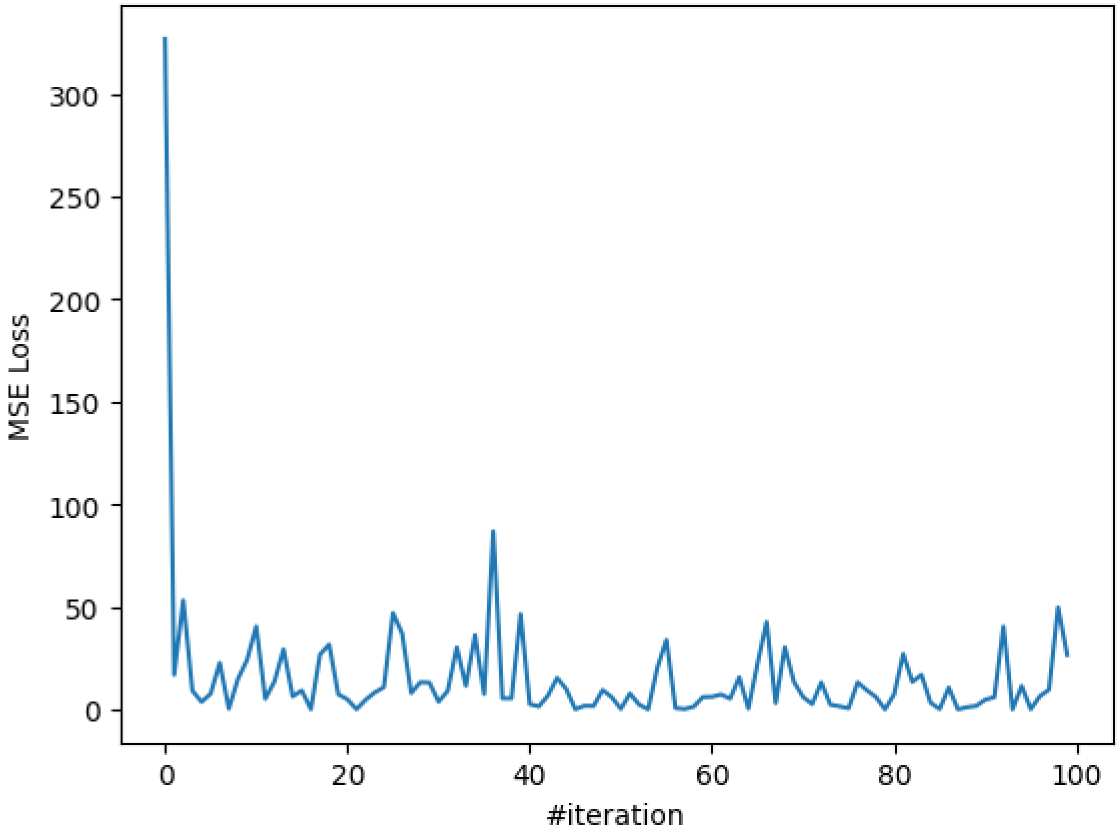
```python
# update weights
def update_weight_wi(wi, dl_dwi, lr):
    wi = wi - lr*dl_dwi
    return wi
```

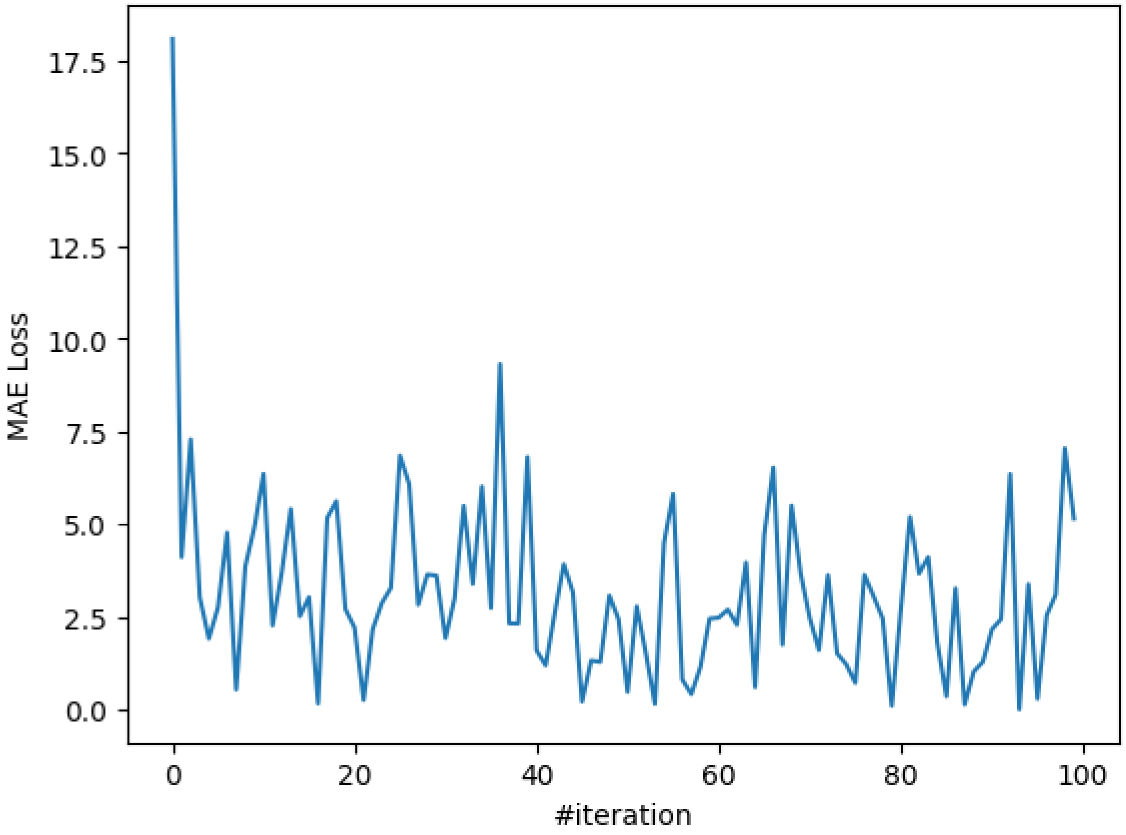
Exercise 2

```
def implement_linear_regression(X_data, y_data, epoch_max = 50,lr = 1e-5):
    losses = []

    w1, w2, w3, b = initialize_params()

    N = len(y_data)
    for epoch in range(epoch_max):
        for i in range(N):
            # get a sample
            x1 = X_data[0][i]
            x2 = X_data[1][i]
            x3 = X_data[2][i]

            y  = y_data[i]

            # print(y)
            # compute output
            y_hat = predict(x1, x2, x3, w1, w2, w3, b)

            # compute loss
            loss = compute_loss_mse(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dw1 = compute_gradient_wi(x1, y, y_hat)
            dl_dw2 = compute_gradient_wi(x2, y, y_hat)
            dl_dw3 = compute_gradient_wi(x3, y, y_hat)
            dl_db  = compute_gradient_b(y, y_hat)

            # update parameters
            w1 = update_weight_wi(w1, dl_dw1, lr)
            w2 = update_weight_wi(w2, dl_dw2, lr)
            w3 = update_weight_wi(w3, dl_dw3, lr)
            b  = update_weight_b(b, dl_db, lr)

            # logging
            losses.append(loss)
    return (w1,w2,w3,b, losses)
```

```
def update_weight_b(b, dl_db, lr):
    b  = b - lr*dl_db
    return b
```

Exercise 2


MSE Loss
300
250
200
150
100
50
0
0
20
40
60
80
100
#iteration


MAE Loss
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20
40
60
80
100
#iteration

Exercise 2

```
# given new data
tv = 19.2
radio = 35.9
newspaper = 51.3

X,y = prepare_data('advertising.csv')
(w1,w2,w3,b, losses) = implement_linear_regression(X,y)
sales = predict(tv, radio, newspaper, w1, w2, w3, b)
print(f'predicted sales is {sales}')
```

Outline

- Linear Regression Review
- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
- Other Discussions

Exercise 3

Bài tập 3 (kỹ thuật huấn luyện data dùng batch N samples - linear regression): Cải tiến giải thuật ở bài tập 2, bằng cách huấn luyện giải thuật linear regression sử dụng N samples-training. Công việc của bạn ở bài tập này là bạn cần implement lại function **implement_linear_regression_nsamples** sử dụng N sample-training với MSE loss function $L(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2$ và MAE loss function $L(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}|\hat{y} - y|$

```
def implement_linear_regression_nsamples(X_data, y_data, epoch_max = 50,lr = 1e-5):
  losses = []

  w1, w2, w3, b = initialize_params()
  N = len(y_data)
```

Exercise 3

```
def implement_linear_regression_nsamples(X_data, y_data, epoch_max = 50,lr = 1e-5):
  losses = []

  w1, w2, w3, b = initialize_params()
  N = len(y_data)

  for epoch in range(epoch_max):

      loss_total = 0.0
      dw1_total = 0.0
      dw2_total = 0.0
      dw3_total = 0.0
      db_total  = 0.0

      for i in range(N):
          # get a sample
          x1 = X_data[0][i]
          x2 = X_data[1][i]
          x3 = X_data[2][i]

          y  = y_data[i]

          # print(y)
          # compute output
          y_hat = predict(x1, x2, x3, w1, w2, w3, b)
```

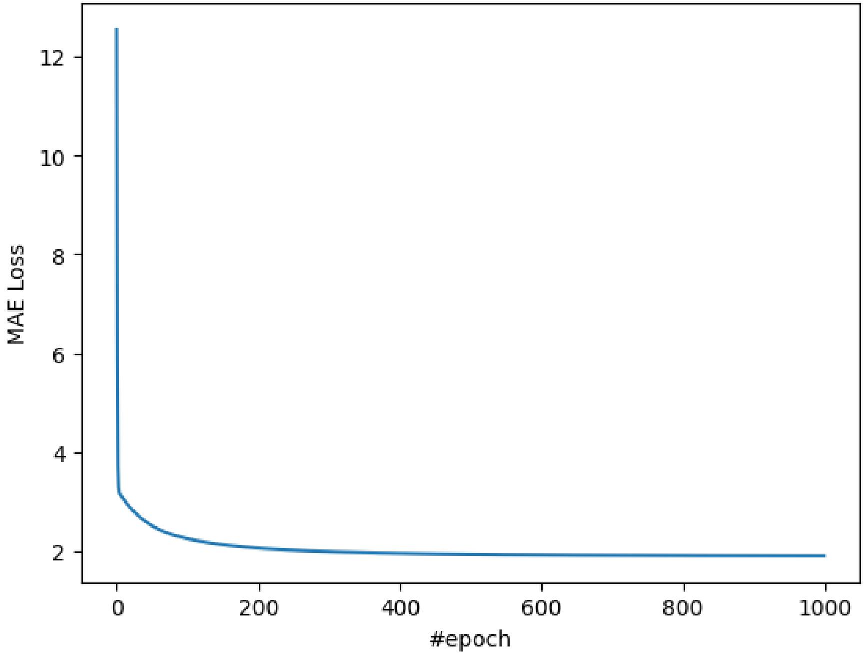
Exercise 3

```python
for i in range(N):
    # get a sample
    x1 = X_data[0][i]
    x2 = X_data[1][i]
    x3 = X_data[2][i]

    y  = y_data[i]

    # print(y)
    # compute output
    y_hat = predict(x1, x2, x3, w1, w2, w3, b)

    # compute loss
    loss = compute_loss_mae(y, y_hat)
    loss_total = loss_total + loss

    # compute gradient w1, w2, w3, b
    dl_dw1 = compute_gradient_wi(x1, y, y_hat)
    dl_dw2 = compute_gradient_wi(x2, y, y_hat)
    dl_dw3 = compute_gradient_wi(x3, y, y_hat)
    dl_db  = compute_gradient_b(y, y_hat)

   # accumulate
    dw1_total = dw1_total + dl_dw1
    dw2_total = dw2_total + dl_dw2
    dw3_total = dw3_total + dl_dw3
    db_total = db_total + dl_db
```

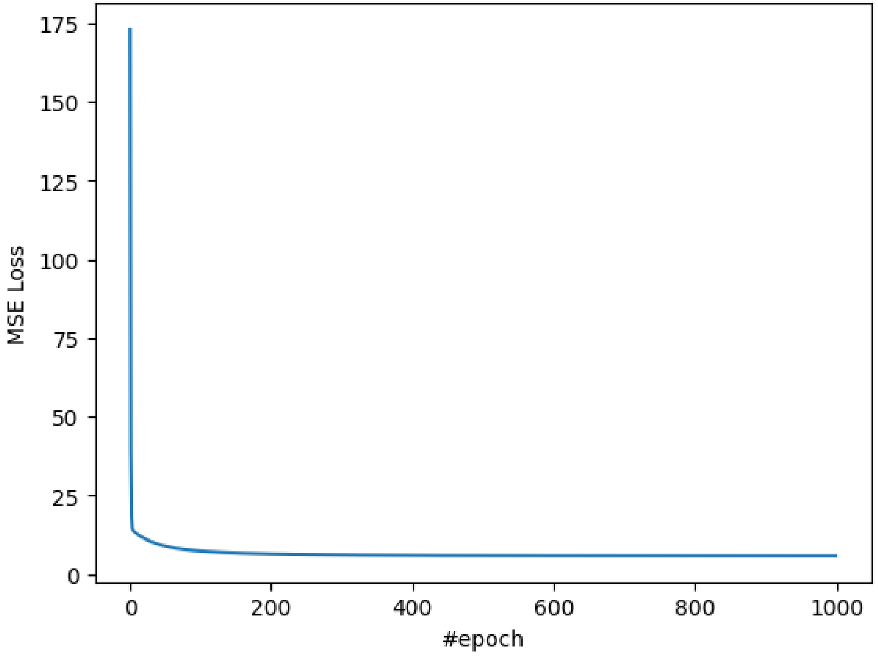
```python
        # (after processing N samples) - update parameters
        w1 = update_weight_wi(w1, dl_dw1/N, lr)
        w2 = update_weight_wi(w2, dl_dw2/N, lr)
        w3 = update_weight_wi(w3, dl_dw3/N, lr)
        b  = update_weight_b(b, dl_db/N, lr)


        # logging
        losses.append(loss_total/N)
    return (w1,w2,w3,b, losses)
```

Exercise 3

```
X,y = prepare_data('advertising.csv')
(w1,w2,w3,b, losses) = implement_linear_regression_nsamples(X,y,1000)
plt.plot(losses)
plt.xlabel("#epoch")
plt.ylabel("MAE Loss")
plt.show()
```




```
X,y = prepare_data('advertising.csv')
(w1,w2,w3,b, losses) = implement_linear_regression_nsamples(X,y,1000)
plt.plot(losses)
plt.xlabel("#epoch")
plt.ylabel("MSE Loss")
plt.show()
```



Outline

- Linear Regression Review
- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
- Other Discussions

Exercise 4

Bài tập 4 Như chúng ta đã biết, mục đích của linear regression là tìm hàm xấp xỉ y = ax1 + bx2 + cx3 + bx0. Trong đó x1 là TV, x2 là Radio, x3 là Newspapers, và x0 = 1. Đầu tiên, bạn cần tổ chức lại dữ liệu đầu vào ở bài tập 1 theo dạng danh sách các feature (x0, x1,x2,x3). Ví dụ theo hình 1, dữ liệu đầu vào dòng thứ 1 và 2 ta có thể tổ chức lại như sau:
X[0] = [1, x1,x2,x3] = [1, 230.1, 37.8, 69.2]
X[1] = [1, x1,x2,x3] = [1, 44.5, 39.3, 45.1]

....
X[199] = [1, x1,x2,x3] = [1, 232.1,8.6, 8,7]
Để implement ý tưởng trên vào chương trình, bạn có thể sử dụng function bên dưới:

Linear Regression

❖ Generalized formula

House price data

Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

Advertising data

Features			Label
TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Model

$$\text{price} = \text{w} * area + b$$

$$\hat{y} = wx + b$$

Model (vectorization)

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} \quad \text{where} \quad \boldsymbol{\theta}^T = [b \quad w]^T$$

$$\boldsymbol{x} = [x_0 \ \ area]^T$$

$$x_0 = 1$$

Model

$$\text{Sale} = w_1 * TV + w_2 * Radio + w_3 * Newspaper + b$$

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Model (vectorization)

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} \quad \text{where} \quad \boldsymbol{\theta}^T = [b \quad w_1 \quad w_2 \quad w_3]^T$$

$$\boldsymbol{x} = [x_0 \ \ TV \quad Radio \quad Newspaper]^T$$

$$x_0 = 1$$

Exercise 4

```
def prepare_data(file_name_dataset):
  data = np.genfromtxt(file_name_dataset, delimiter=',', skip_header=1).tolist()

  # get tv (index=0)
  tv_data = get_column(data, 0)

  # get radio (index=1)
  radio_data = get_column(data, 1)

  # get newspaper (index=2)
  newspaper_data = get_column(data, 2)

  # get sales (index=3)
  sales_data = get_column(data, 3)

  # building X input  and y output for training
  #Create list of features for input
  X = [[1, x1, x2, x3] for x1, x2, x3 in zip(tv_data, radio_data, newspaper_data)]
  y = sales_data
  return X,y
```

AI VIET NAM
@aivietnam.edu.vn

Exercise 4

```
def implement_linear_regression(X_feature, y_ouput, epoch_max = 50,lr = 1e-5):

    losses = []
    weights = initialize_params()
    N = len(y_ouput)
    for epoch in range(epoch_max):
        print("epoch", epoch)
        for i in range(N):
            # get a sample - row i
            features_i = X_feature[i]
            y = sales_data[i]

            # compute output
            y_hat = predict(features_i, weights)

            # compute loss
            loss = compute_loss(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dweights = compute_gradient_w(features_i, y, y_hat)

            # update parameters
            weights = update_weight(weights, dl_dweights, lr)

            # logging
            losses.append(loss)
    return weights, losses
```

```
def predict(X_features, weights):
    return sum([f*w for f, w in zip(X_features, weights)])
```

Exercise 4

```python
def implement_linear_regression(X_feature, y_ouput, epoch_max = 50,lr = 1e-5):

    losses = []
    weights = initialize_params()
    N = len(y_ouput)
    for epoch in range(epoch_max):
        print("epoch", epoch)
        for i in range(N):
            # get a sample - row i
            features_i = X_feature[i]
            y = sales_data[i]

            # compute output
            y_hat = predict(features_i, weights)

            # compute loss
            loss = compute_loss(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dweights = compute_gradient_w(features_i, y, y_hat)

            # update parameters
            weights = update_weight(weights, dl_dweights, lr)

            # logging
            losses.append(loss)
    return weights, losses
```

```python
def compute_loss(y_hat, y):
    return (y_hat - y)**2
```

Exercise 4

```
def implement_linear_regression(X_feature, y_ouput, epoch_max = 50,lr = 1e-5):

    losses = []
    weights = initialize_params()
    N = len(y_ouput)
    for epoch in range(epoch_max):
        print("epoch", epoch)
        for i in range(N):
            # get a sample - row i
            features_i = X_feature[i]
            y = sales_data[i]

            # compute output
            y_hat = predict(features_i, weights)

            # compute loss
            loss = compute_loss(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dweights = compute_gradient_w(features_i, y, y_hat)

            # update parameters
            weights = update_weight(weights, dl_dweights, lr)

            # logging
            losses.append(loss)
    return weights, losses
```

```
# compute gradient
def compute_gradient_w(X_features, y, y_hat):
    dl_dweights = [2*xi*(y_hat-y) for xi in X_features]
    return dl_dweights
```

AI VIET NAM
@aivietnam.edu.vn

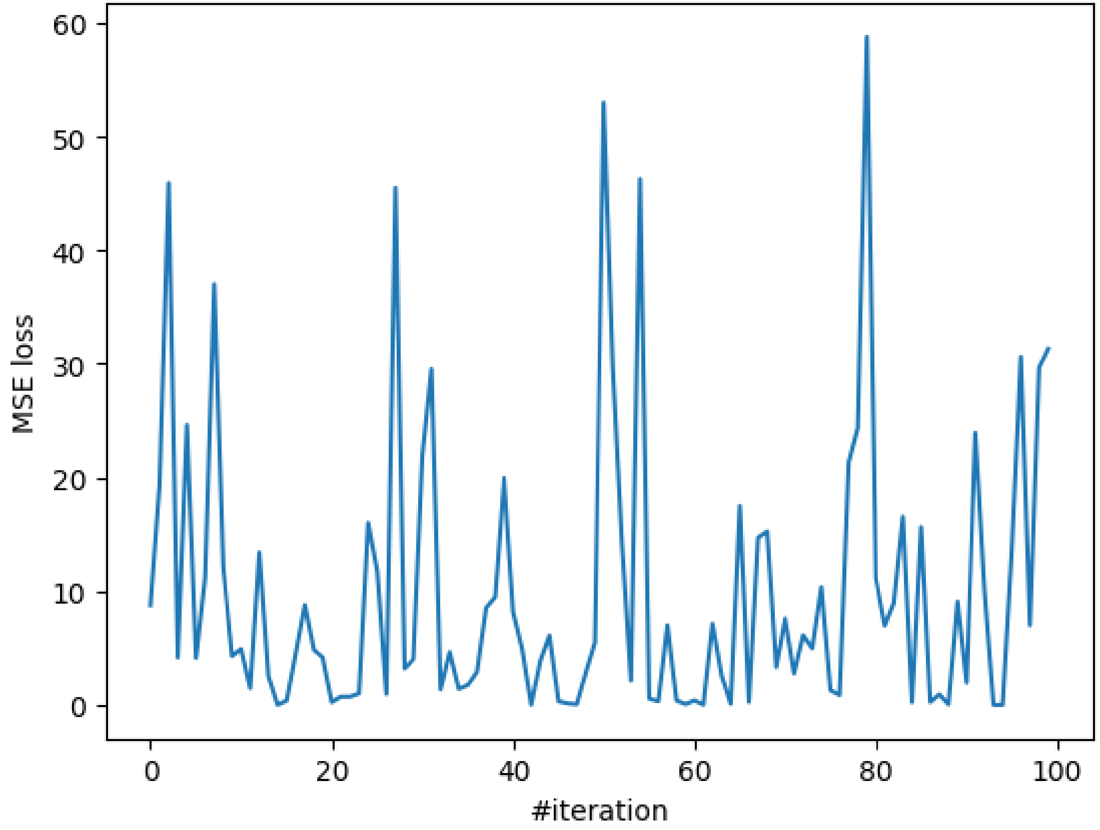
Exercise 4

```
def implement_linear_regression(X_feature, y_ouput, epoch_max = 50,lr = 1e-5):

    losses = []
    weights = initialize_params()
    N = len(y_ouput)
    for epoch in range(epoch_max):
        print("epoch", epoch)
        for i in range(N):
            # get a sample - row i
            features_i = X_feature[i]
            y = sales_data[i]

            # compute output
            y_hat = predict(features_i, weights)

            # compute loss
            loss = compute_loss(y, y_hat)

            # compute gradient w1, w2, w3, b
            dl_dweights = compute_gradient_w(features_i, y, y_hat)

            # update parameters
            weights = update_weight(weights, dl_dweights, lr)

            # logging
            losses.append(loss)
    return weights, losses
```

```
# update weights
def update_weight(weights, dl_dweights, lr):
    weights = [w - lr*dw for w, dw in zip(weights, dl_dweights)]
    return weights
```

Exercise 4

```
X,y = prepare_data('advertising.csv')
W,L = implement_linear_regression(X,y)
plt.plot(L[-100:])
plt.xlabel("#iteration")
plt.ylabel("MSE loss")
plt.show()
```



Outline

- Linear Regression Review
- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4


- Exercise 5
- Other Discussions

Exercise 5

Bài tập 5 (Tìm hiểu kỹ thuật Feature Scaling thông qua min and max)): Ở bài tập này các bạn cần

phải chuẩn hoá dữ liệu đầu vào X trước thuật nhằm tăng tốc độ hội tụ của giải thuật linear regression. Yêu cầu của bài tập là các bạn cần chuẩn hoá data theo MinMax scale $X_{new} = \frac{X_{old} - X_{min}}{X_{max} - X_{min}}$ trước khi đưa vào huấn luyện data. Nhiệm vụ của bạn ở bài tập này là cần implement hàm **min_max_scaling()** để chuẩn hoá dữ liệu input X như bên dưới:

AI VIET NAM
@aivietnam.edu.vn

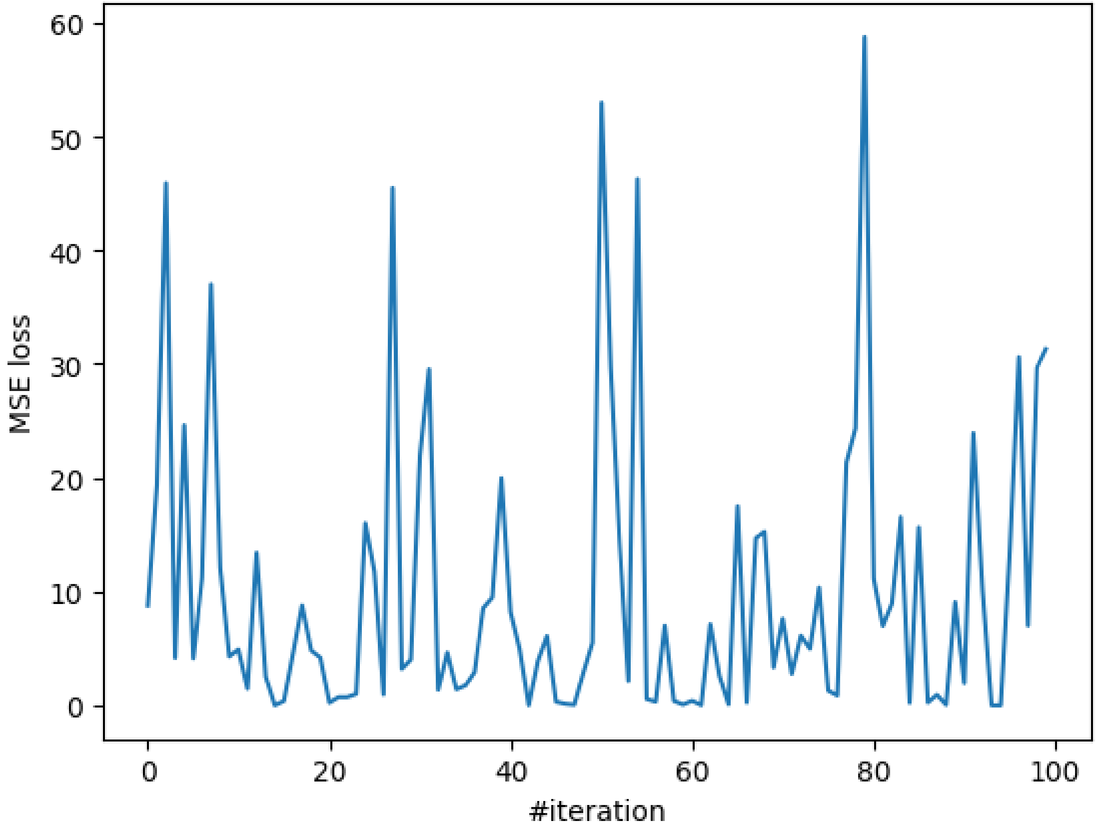
Exercise 5

```
def prepare_data(file_name_dataset):
  data = np.genfromtxt(file_name_dataset, delimiter=',', skip_header=1).tolist()

  # get tv (index=0)
  tv_data = get_column(data, 0)

  # get radio (index=1)
  radio_data = get_column(data, 1)

  # get newspaper (index=2)
  newspaper_data = get_column(data, 2)

  # get sales (index=3)
  sales_data = get_column(data, 3)

  # scale data (only for features)
  # remember to scale input features in inference, therefore, we need to save max, min and mean values
  (tv_data, radio_data, newspaper_data), (max_data_1, max_data_2, max_data_3, min_data_1, min_data_2, min_data_3)
  = min_max_scaling(tv_data,radio_data,newspaper_data)

  # building X input  and y output for training
  #Create list of features for input
  X = [[1, x1, x2, x3] for x1, x2, x3 in zip(tv_data, radio_data, newspaper_data)]
  y = sales_data
  return X,y
```

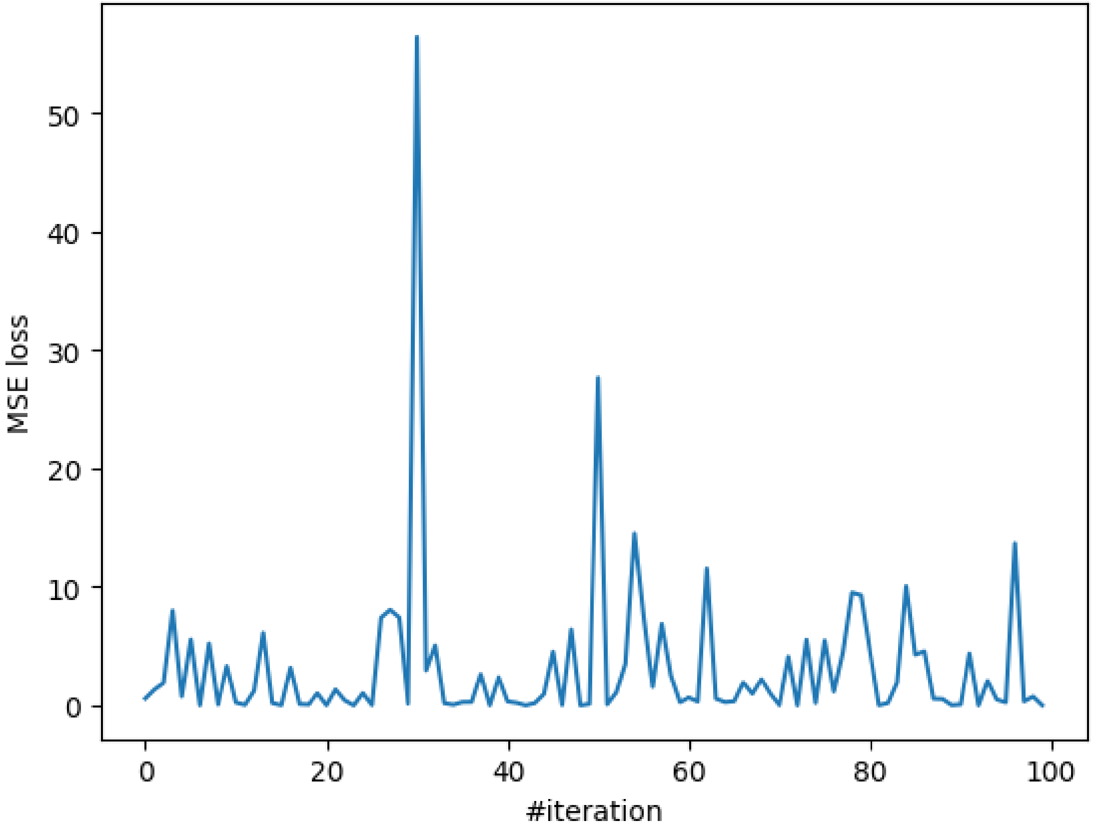
Exercise 5

```python
def min_max_scaling(data1, data2, data3):
    max_data_1 = max(data1)
    max_data_2 = max(data2)
    max_data_3 = max(data3)

    min_data_1 = min(data1)
    min_data_2 = min(data2)
    min_data_3 = min(data3)



    data1 = [(x - min_data_1) / (max_data_1 - min_data_1) for x in data1]
    data2 = [(x - min_data_2) / (max_data_2 - min_data_2) for x in data2]
    data3 = [(x - min_data_3) / (max_data_3 - min_data_3) for x in data3]

    return (data1, data2, data3), (max_data_1, max_data_2, max_data_3, min_data_1, min_data_2, min_data_3)
```

Exercise 5



Before Feature Scaling



After Feature Scaling

Outline

- Linear Regression Review
- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
-  Other Discussions

How to measure the accurracy of the system

X　　　　　　　　　　　　　　　　　　　　　Y

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9
8.7	48.9	75	7.2
57.5	32.8	23.5	11.8
120.2	19.6	11.6	13.2
8.6	2.1	1	4.8
199.8	2.6	21.2	15.6
66.1	5.8	24.2	12.6
214.7	24	4	17.4
23.8	35.1	65.9	9.2
97.5	7.6	7.2	13.7
204.1	32.9	46	19
195.4	47.7	52.9	22.4
67.8	36.6	114	12.5
281.4	39.6	55.8	24.4
69.2	20.5	18.3	11.3
147.3	23.9	19.1	14.6
218.4	27.7	53.4	18
237.4	5.1	23.5	17.5

Training

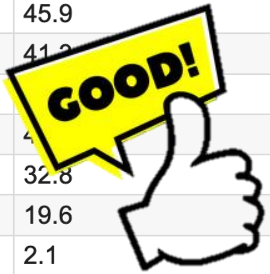
Strategy:

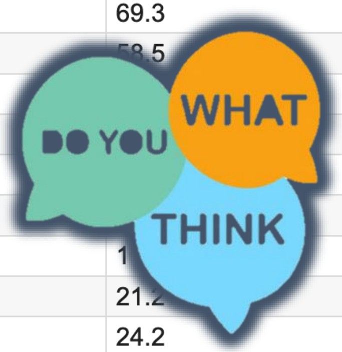
- Train: 200 samples

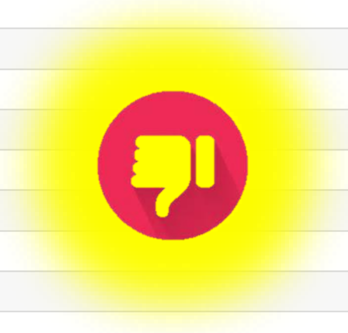
How to measure the accurracy of the system

X Y

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	[illegible]	[illegible]	17.9
8.7	[illegible]	[illegible]	7.2
57.5	32.8	[illegible]	11.8
120.2	19.6	[illegible]	13.2
8.6	2.1	[illegible]	4.8
199.8	2.6	21.2	15.6
66.1	5.8	24.2	12.6
214.7	24	4	17.4
23.8	35.1	65.9	9.2
97.5	7.6	7.2	13.7
204.1	32.9	46	19
195.4	47.7	52.9	22.4
67.8	36.6	114	12.5
281.4	39.6	55.8	24.4
69.2	20.5	18.3	11.3
147.3	23.9	19.1	14.6
218.4	27.7	53.4	18
237.4	5.1	23.5	17.5







Testing

Strategy: n = 200
- Train: 200 samples
- Test: 200 samples

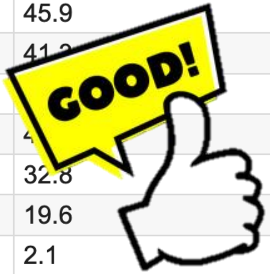
```
100% dataset for
training
100% dataset for
```

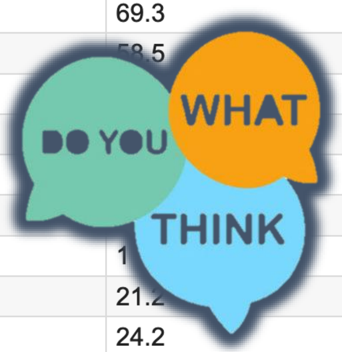
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

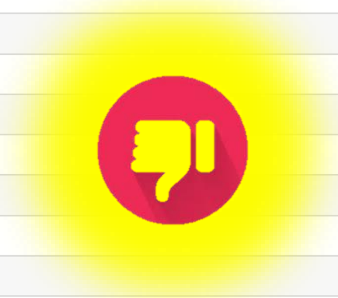
How to measure the accurracy of the system

$$\overbrace{\text{TV} \quad \text{Radio} \quad \text{Newspaper}}^{X} \qquad \overbrace{\text{Sales}}^{Y}$$

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	[illegible]	[illegible]	17.9
8.7	[illegible]	[illegible]	7.2
57.5	32.8	[illegible]	11.8
120.2	19.6	[illegible]	13.2
8.6	2.1	[illegible]	4.8
199.8	2.6	21.2	15.6
66.1	5.8	24.2	12.6
214.7	24	4	17.4
23.8	35.1	65.9	9.2
97.5	7.6	7.2	13.7
204.1	32.9	46	19
195.4	47.7	52.9	22.4
67.8	36.6	114	12.5
281.4	39.6	55.8	24.4
69.2	20.5	18.3	11.3
147.3	23.9	19.1	14.6
218.4	27.7	53.4	18
237.4	5.1	23.5	17.5







Testing

Strategy:
- Train: 160 samples
- Test: 40 samples

80% dataset for training
20% dataset for testing
Train set ≠ Test set

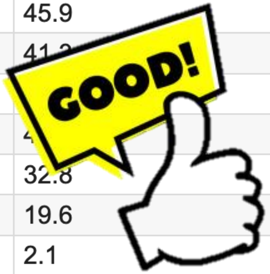
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

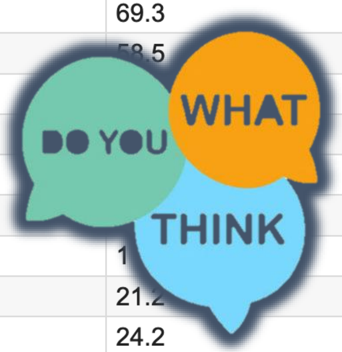
How to measure the accurracy of the system

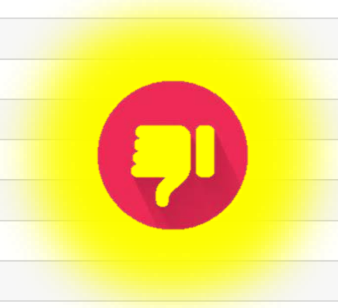
X

Y

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	[illegible]	[illegible]	17.9
8.7	4[illegible]	[illegible]	7.2
57.5	32.8	[illegible]	11.8
120.2	19.6	[illegible]	13.2
8.6	2.1	1[illegible]	4.8
199.8	2.6	21.2	15.6
66.1	5.8	24.2	12.6
214.7	24	4	17.4
23.8	35.1	65.9	9.2
97.5	7.6	7.2	13.7
204.1	32.9	46	19
195.4	47.7	52.9	22.4
67.8	36.6	114	12.5
281.4	39.6	55.8	24.4
69.2	20.5	18.3	11.3
147.3	23.9	19.1	14.6
218.4	27.7	53.4	18
237.4	5.1	23.5	17.5







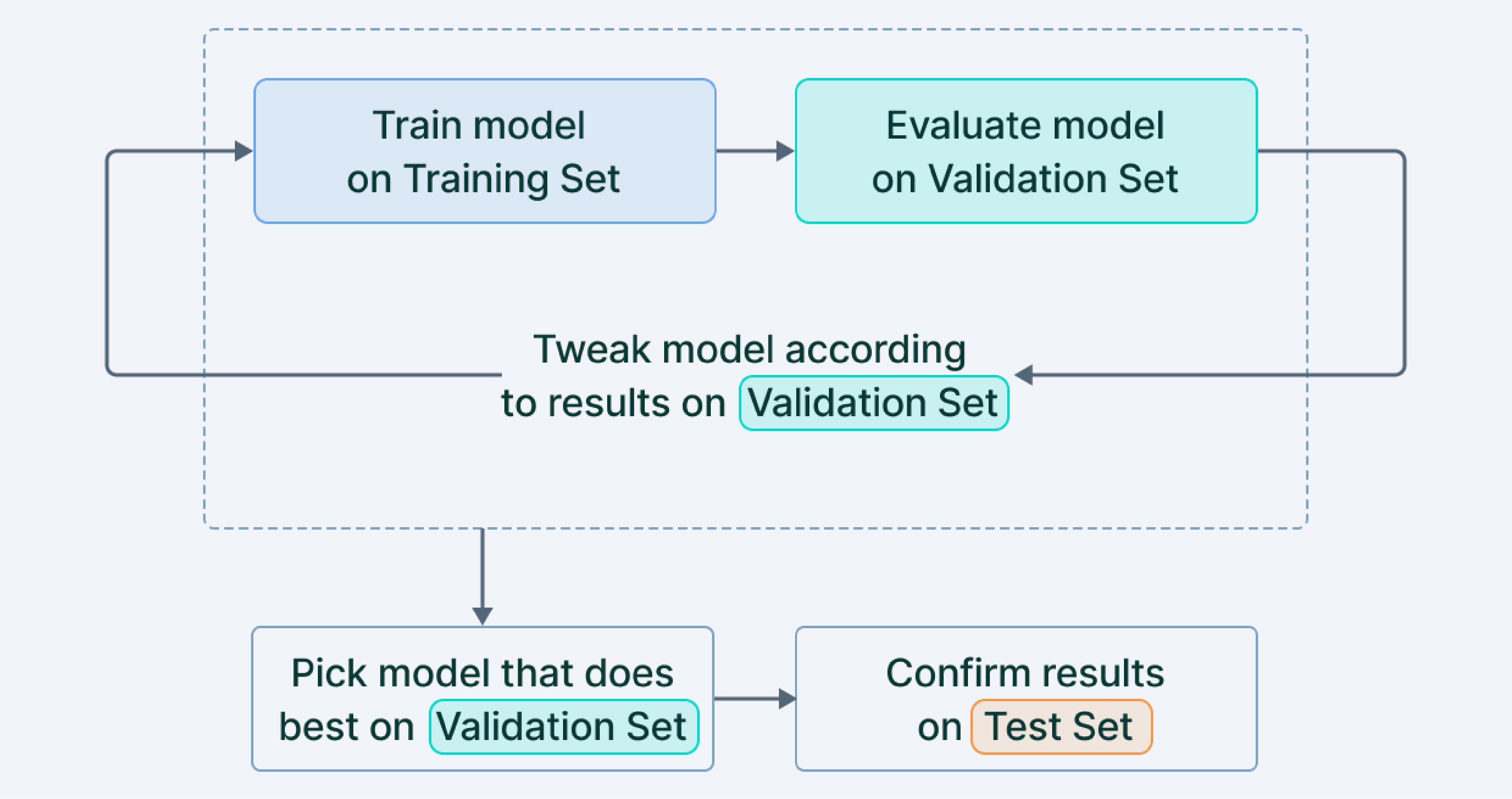
Testing

Strategy:
- Train: 160 samples
- Test: 40 samples

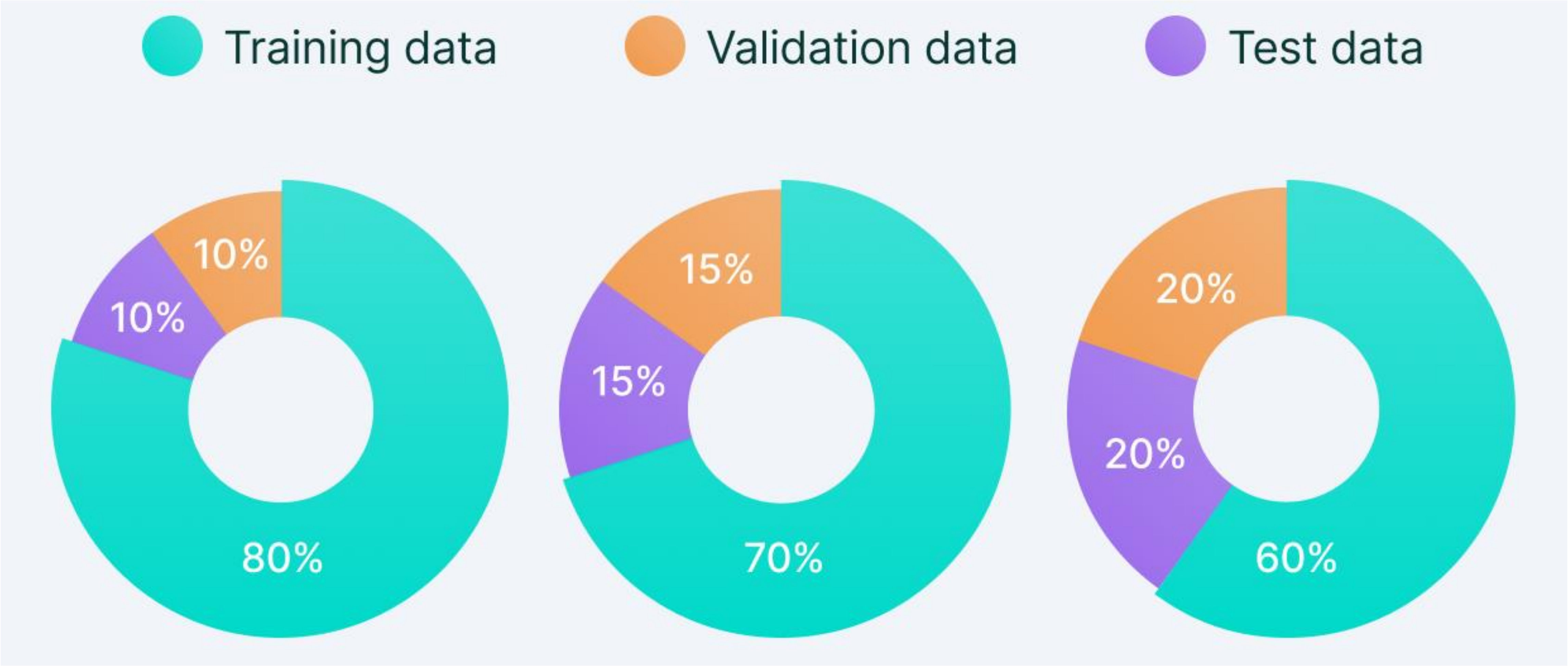
60% dataset for training
20% dataset for

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

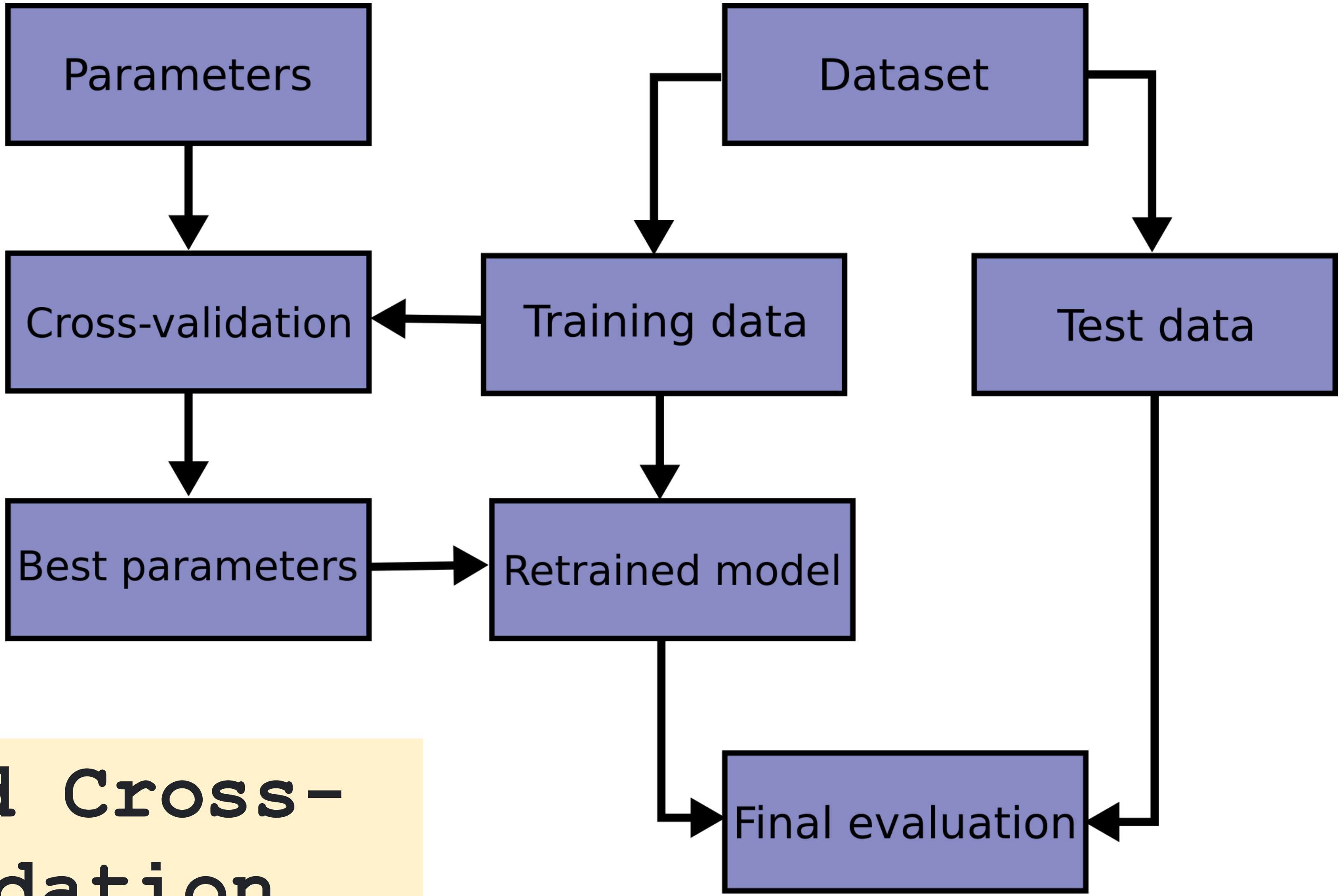
Train/Validation/Test



Exercise 5


Training data
Validation data
Test data
10%
10%
80%
15%
15%
70%
20%
20%
60%

Exercise 5



K Fold Cross-validation

AI VIET NAM
@aivietnam.edu.vn

Exercise 5