AI VIETNAM
All-in-One Course

Numpy for Vector and Matrix Operations

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

References/Reading

NumPy user guide

This guide is an overview and explains the important features

- What is NumPy?
- Installation
- NumPy quickstart
- NumPy: the absolute basics for beginners
- NumPy fundamentals
- Miscellaneous
- NumPy for MATLAB users
- Building from source
- Using NumPy C-API
- NumPy Tutorials
- NumPy How Tos
- For downstream package authors

https://numpy.org/doc/stable/user/index.html

Chapter 2

Matrices

2.1 Operations with Matrices

Homework: §2.1 (page 56): 7, 9, 13, 15, 17, 25, 27, 35, 37, 41, *46*, 49, 67

Main points in this section:

1. *We define a few concept regarding matrices. This would include addition of matrices, scalar multiplication and multiplication of matrices.*
2. *We also represent a system of linear equation as equation with matrices.*

https://mandal.ku.edu/math290/m290NotesChap2.pdf

Objective

❖ Traditional notation → Vectorized one

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ x \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} b \\ w \end{bmatrix}$$

Tradition

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

Vectorization

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

2) Compute output $\hat{y}$

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{\theta}$$

3) Compute loss

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

4) Compute derivative

$$L'_{\boldsymbol{\theta}} = 2\boldsymbol{x}(\hat{y} - y)$$

5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta L'_{\boldsymbol{\theta}}$$

η is learning rate

Objective

❖ Implementation (tradition)

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

```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)
```

Objective

❖ Implementation (vectorization using numpy)

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

2) Compute output $\hat{y}$

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{\theta}$$

3) Compute loss

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

4) Compute derivative

$$L'_{\boldsymbol{\theta}} = 2\boldsymbol{x}(\hat{y} - y)$$

5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta L'_{\boldsymbol{\theta}}$$

η is learning rate

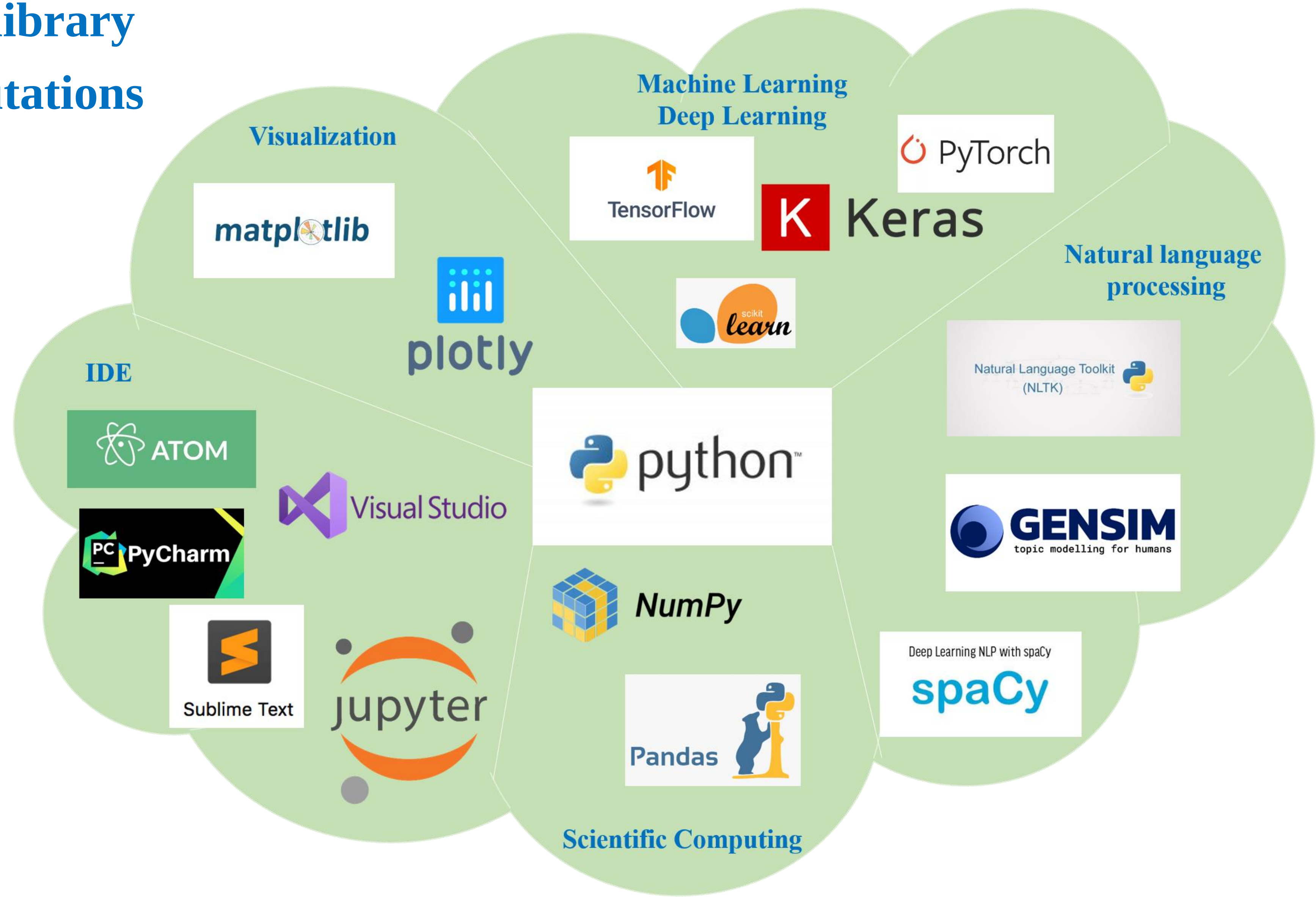
```python
import numpy as np

# forward
def predict(x, theta):
    return x.dot(theta)

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

    return dtheta

# update weights
def update_weight(theta, lr, dtheta):
    dtheta_new = theta - lr*dtheta

    return dtheta_new
```

Outline

- Introduction to Numpy
- Numpy Examples
- Introduction to Vector and Matrix
- Vectorize the Linear Regression (1-sample)

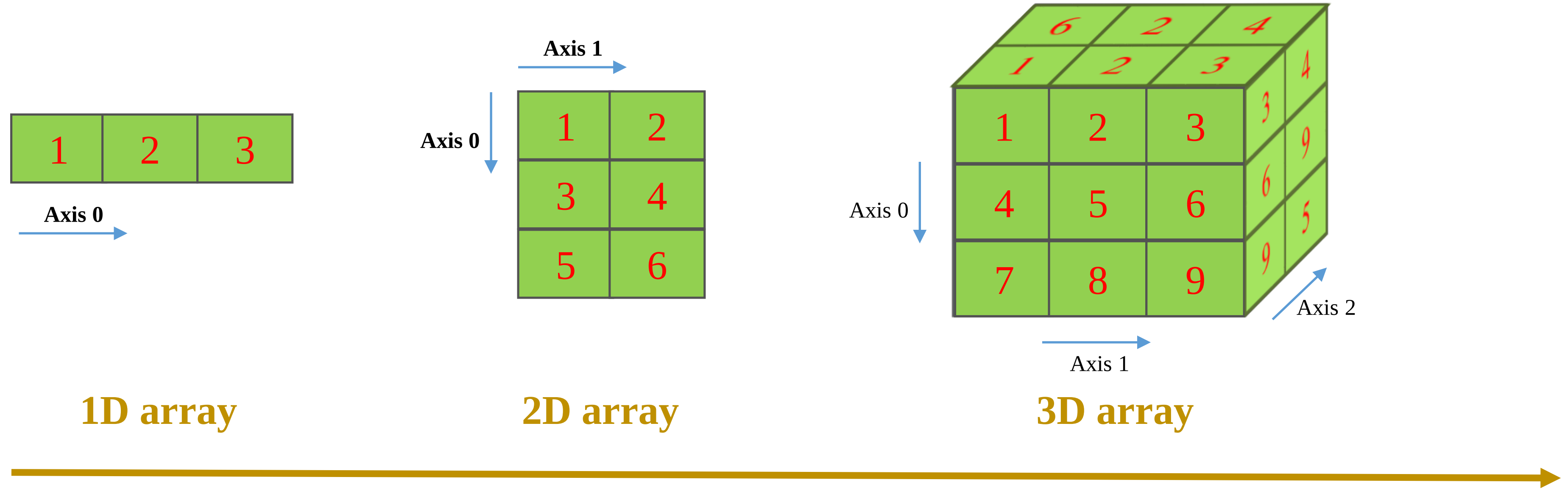
Introduction to Numpy

- ❖ **Numpy is a Python library**
- ❖ **For scientific computations**



Introduction to Numpy

- Numpy is a Python library
- For scientific computations
- Numpy array ⟺ Tensor in Tensorflow and Pytorch
- Numpy arrays are multi-dimensional arrays

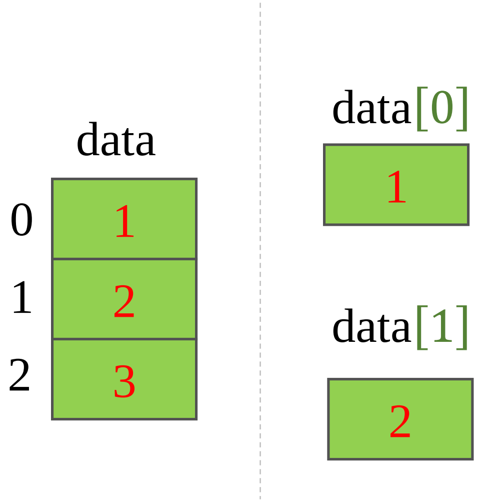


Introduction to Numpy

❖ Create Numpy array

❖ From List

arr_np = np.array(python_list)



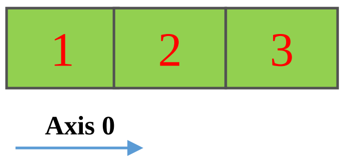

```python
# tạo ndarray từ list

import numpy as np

# tạo list
l = list(range(1, 4))

# tạo ndarray
data = np.array(l)

print(data)
print(data[0])
print(data[1])
```

```
[1 2 3]
1
2
```

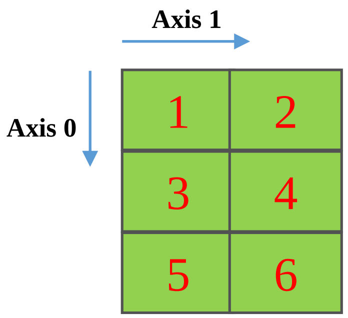
Introduction to Numpy

❖ Common attributes

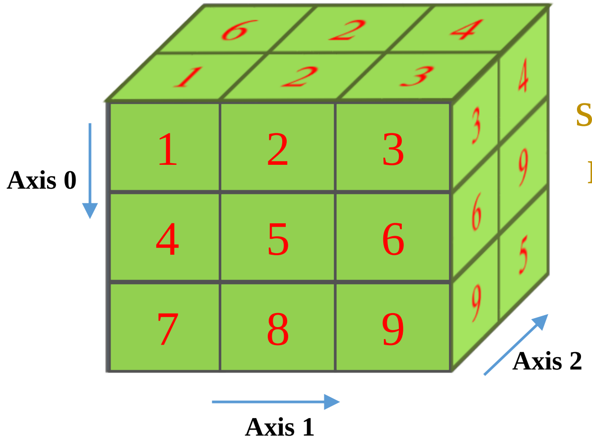
- ❖ dtype: data type
- ❖ shape: return a tuple of #elements in each dimension
- ❖ ndim: return #dimensions



shape=(3,)
ndim=1



shape=(3,2)
ndim=2



shape=(3,3,2)
ndim=2

```python
# tạo ndarray từ list

import numpy as np

# tạo list
list1D = [1,2,3]

# tạo ndarray
data = np.array(list1D)

print(data)
print(data.shape)
```

```
[1 2 3]
(3,)
```

```python
# tạo ndarray từ list

import numpy as np

# tạo list
list2D = [[1,2],[3,4],[5,6]]

# tạo ndarray
data = np.array(list2D)

print(data)
print(data.shape)
```

```
[[1 2]
 [3 4]
 [5 6]]
(3, 2)
```

```python
# tạo ndarray từ list

import numpy as np

# tạo list
list3D = [[[1,6], [2,2], [3,4]],
          [[4,7], [5,2], [6,9]],
          [[7,7], [8,2], [9,5]]]

# tạo ndarray
data = np.array(list3D)

#print(data)
print(data.shape)
```

```
(3, 3, 2)
```

Introduction to Numpy

❖ Common attributes

- ❖ dtype: data type
- ❖ shape: return a tuple of #elements in each dimension
- ❖ ndim: return #dimensions

dtype example

```
import numpy as np

# tạo ndarray
data1 = np.array([1,2,3])
print(data1.dtype)

data2 = np.array([1.,2.,3.])
print(data2.dtype)

data3 = np.array([1,2,3], dtype=np.int64)
print(data3.dtype)
```

```
int32
float64
int64
```

ndim example

```
import numpy as np

# tạo ndarray
data1 = np.array([1,2,3])
print(data1.ndim)

data2 = np.array([[1,2,3]])
print(data2.ndim)

data3 = np.array([[1],[2],[3]])
print(data3.ndim)
```

```
1
2
2
```

Introduction to Numpy

❖ Create Numpy arrays

zeros() function

	0	1	2
0	0	0	0
1	0	0	0

```python
# Tạo một numpy array
# với tất cả phần tử là 0

import numpy as np

# shape: 2 dòng, 3 cột
arr = np.zeros((2,3))
print(arr)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

ones() function

	0	1	2
0	1	1	1
1	1	1	1

```python
# Tạo một numpy array với
# tất cả phần tử là 1

import numpy as np

# numpy.ones(shape)
# shape: 2 dòng, 3 cột
arr = np.ones((2,3))
print(arr)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

full() function

	0	1	2
0	9	9	9
1	9	9	9

```python
# Tạo một numpy array với tất
# cả phần tử là hằng số fill_value

import numpy as np

# numpy.full(shape, fill_value)
# shape: 2 dòng, 3 cột
arr = np.full((2,3), 9)
print(arr)
```

```
[[9 9 9]
 [9 9 9]]
```

Introduction to Numpy

❖ Create Numpy arrays

arange() function

	0	1	2	3	4
arr1 =	0	1	2	3	4

	0	1	2
arr2 =	0	2	4

```python
import numpy as np

# np.arange(start=0, stop, step=1)
arr1 = np.arange(5)
print(arr1)

arr2 = np.arange(0, 5, 2)
print(arr2)
```

```
[0 1 2 3 4]
[0 2 4]
```

eye() function

	0	1	2
0	1	0	0
1	0	1	0
2	0	0	1

```python
# Tạo một numpy array với đường chéo là số 1
# số 0 được điền vào những ô phần tử còn lại

import numpy as np

# numpy.eye(N)
# shape: 3 dòng, 3 cột
arr = np.eye(3)
print(arr)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

random() function

	0	1	2
0	0.574	0.682	0.704
1	0.806	0.844	0.799

```python
# Tạo một numpy array với
# giá trị ngẫu nhiên

import numpy as np

# np.random.random(size)
# shape: 2 dòng, 3 cột; với
# phần tử có giá trị ngẫu nhiên
arr = np.random.random((2,3))
print(arr)
```

```
[[0.57488062 0.68266312 0.70438569]
 [0.80661973 0.84413356 0.79905247]]
```

Introduction to Numpy

❖ Slicing

arr[for_axis_0, for_axis_1, …]

‘:’ - get all the elements

‘start : stop : step’

get the elements from a^{th} to $(b^{th}-1)$

Could be ignored

step = 1

data =

0	1	2	3	4	5
1	2	3	4	5	6

data[::1] =

0	1	2	3	4	5
1	2	3	4	5	6

data[2:5:1] =

0	1	2
3	4	5

data[::-1] =

0	1	2	3	4	5
6	5	4	3	2	1

data[2:5:-1] =

0	1	2
6	5	4

data[:] =

0	1	2	3	4	5
1	2	3	4	5	6

data[:3] =

0	1	2
1	2	3

Introduction to Numpy

❖ **Slicing**

arr[for_axis_0, for_axis_1, …]

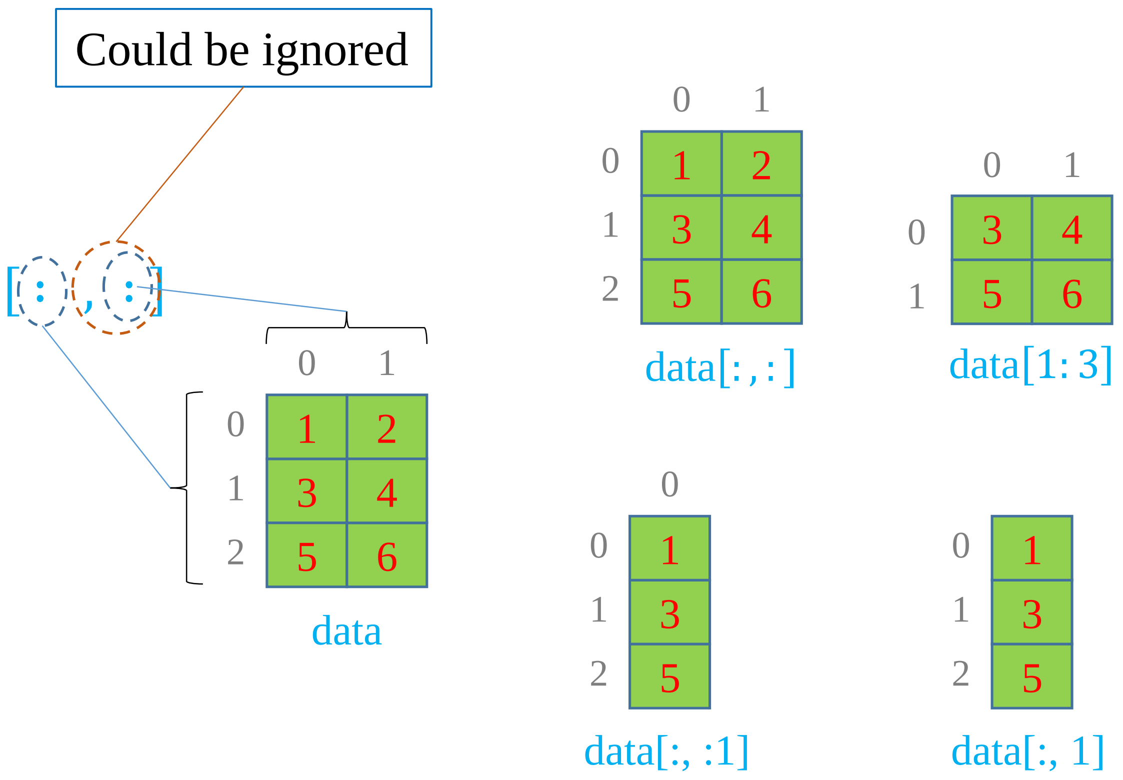
‘:’ - get all the elements

‘start : stop : step’

get the elements from ath to (bth−1)

Could be ignored

step = 1



	Feature	Label
	area	price
0	6.7	9.1
1	4.6	5.9
2	3.5	4.6
3	5.5	6.7
	0	1

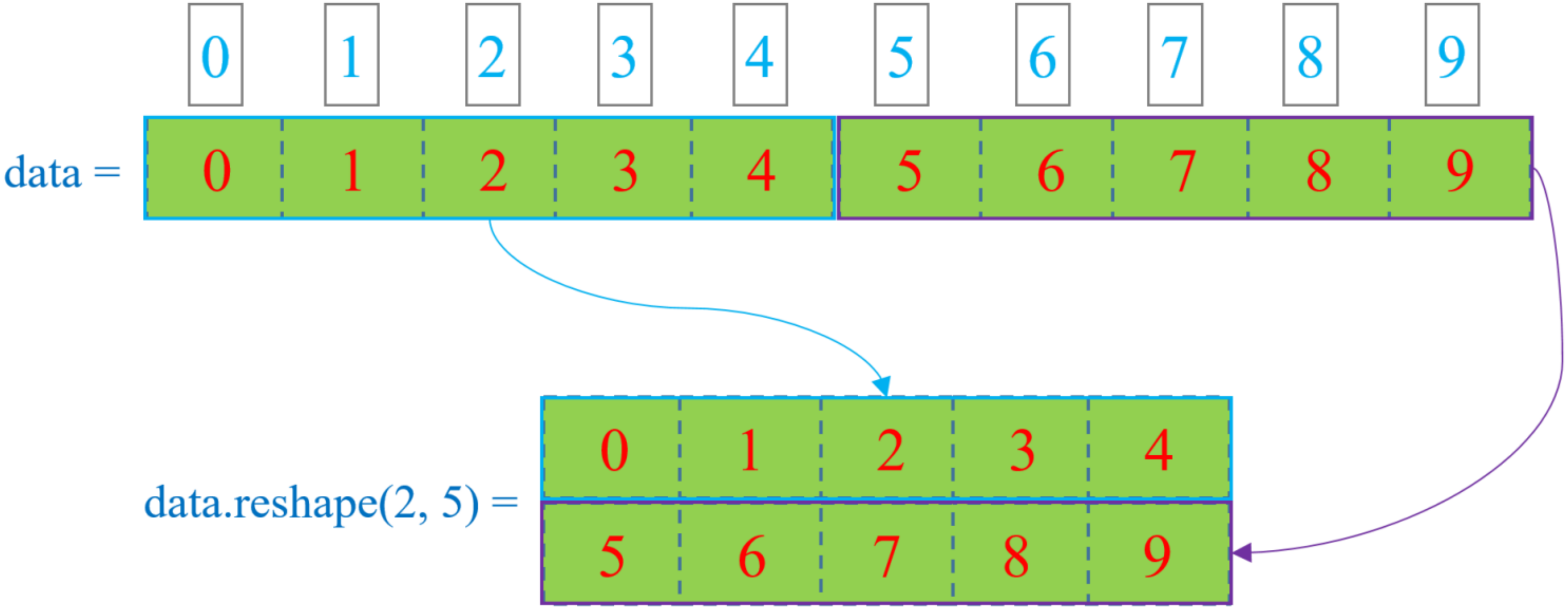
Get the area column as a matrix?

Get the price column as a vector?

❖ Reshape an array

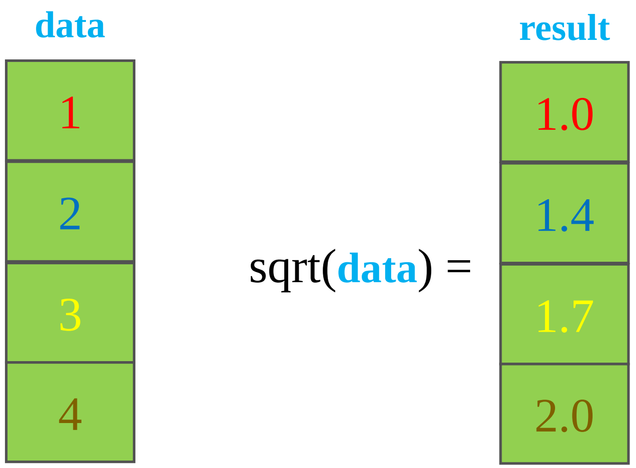
```python
# Python code
import numpy as np

# create a 1D ndarray from 0 to 9
data = np.arange(10)
print(data)

# reshape data to 2 rows and 5 columns
data_2d = data.reshape(2, 5)
print(data_2d)
```

```
================= Output =================
[0 1 2 3 4 5 6 7 8 9]
[[0 1 2 3 4]
 [5 6 7 8 9]]
==========================================
```



Numpy

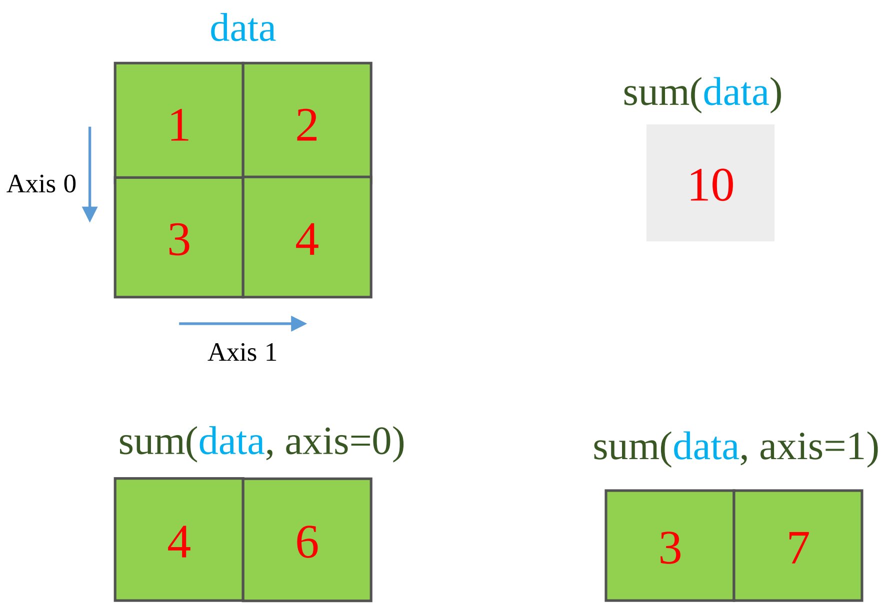
Square root




```python
data = np.array([1,2,3,4])

print('data \n', data)

# Căn bậc 2 từng phần tử trong data
print('sqrt \n', np.sqrt(data))
```

```
data
 [1 2 3 4]
sqrt
 [1.         1.41421356 1.73205081 2.        ]
```

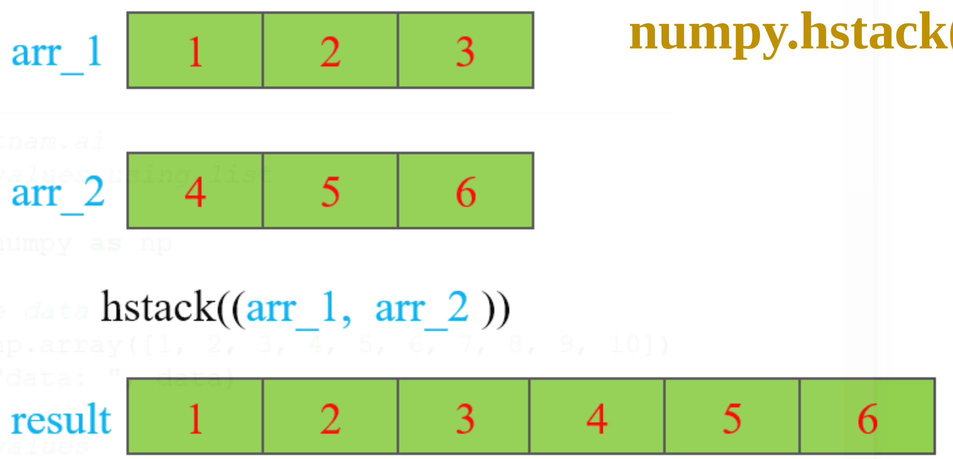
Summation




```python
data = np.array([[1,2],
                 [3,4]])

# Tổng các phần tử của mảng
print(np.sum(data))

# Tính tổng theo từng cột
print(np.sum(data, axis=0))

# Tính tổng theo từng dòng
print(np.sum(data, axis=1))
```

```
10
[4 6]
[3 7]
```

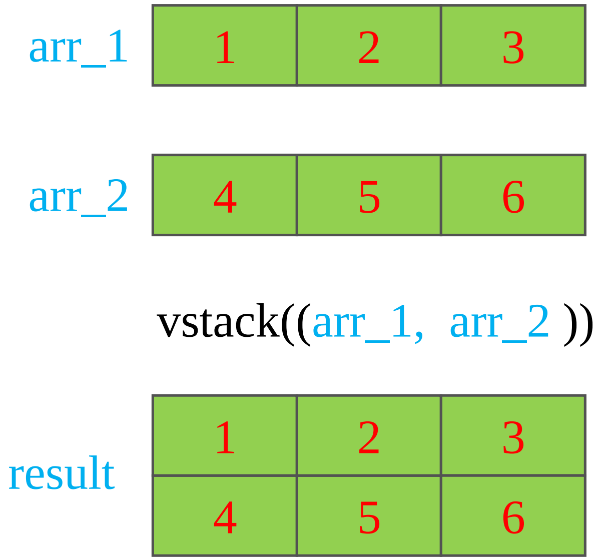
numpy.hstack()




```python
import numpy as np

arr_1 = np.array([1, 2, 3])
print("arr_1: ", arr_1)

arr_2 = np.array([4, 5, 6])
print("arr_2: ", arr_2)

# kết hợp array theo chiều ngang
result = np.hstack((arr_1, arr_2))
print("result: ", result)
```

```
arr_1:  [1 2 3]
arr_2:  [4 5 6]
result:  [1 2 3 4 5 6]
```

numpy.vstack()

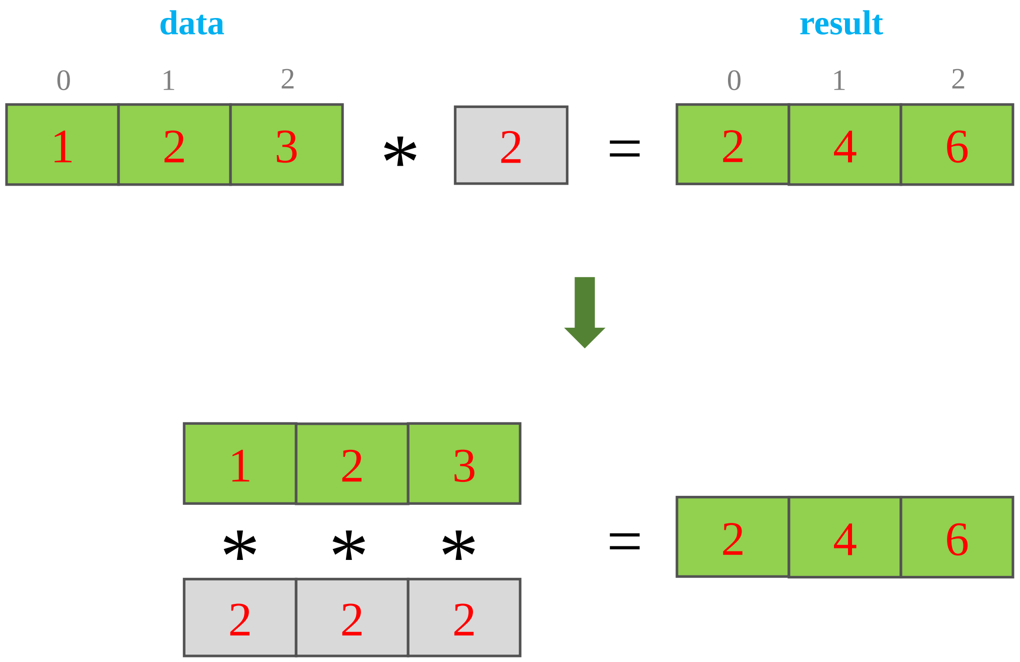



```python
arr_1 = np.array([1, 2, 3])
print("arr_1: ", arr_1)

arr_2 = np.array([4, 5, 6])
print("arr_2: ", arr_2)

# kết hợp array theo chiều dọc
result = np.vstack((arr_1, arr_2))
print("result: \n", result)
```

```
arr_1:  [1 2 3]
arr_2:  [4 5 6]
result:
 [[1 2 3]
 [4 5 6]]
```

Broadcasting

❖ Vector and a scalar




```python
# aivietnam.ai
import numpy as np

# create data
data = np.array([1,2,3])
factor = 2

# broadcasting
result_multiplication = data*factor
result_minus = data - factor

print(data)
print(result_multiplication)
print(result_minus)
```

```
[1 2 3]
[2 4 6]
[-1  0  1]
```

Broadcasting

❖ Matrix and vector

$$
X = \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 7 & 8 & 9 \\ 3 & 10 & 11 & 12 \end{array}
\quad + \quad
v = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}
$$

$$\Downarrow$$

$$
\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 7 & 8 & 9 \\ 3 & 10 & 11 & 12 \end{array}
\quad + \quad
\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}
\quad = \quad
Y = \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 2 & 2 & 4 \\ 1 & 5 & 5 & 7 \\ 2 & 8 & 8 & 10 \\ 3 & 11 & 11 & 13 \end{array}
$$

```python
# aivietnam.ai
import numpy as np

X = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9],
              [10, 11, 12]])
v = np.array([1, 0, 1])

Y = X + v
print(Y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

Outline

- Introduction to Numpy
- Numpy Examples
- Introduction to Vector and Matrix
- Vectorize the Linear Regression (1-sample)

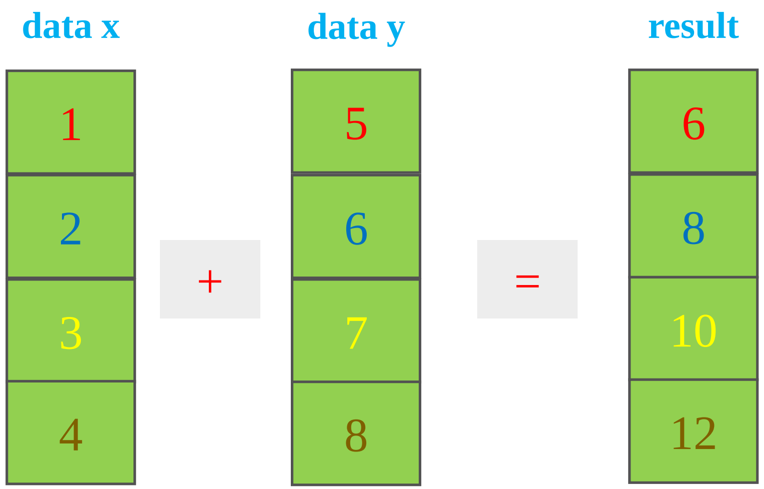
Vector & Matrix

Vector	Matrix
n is a natural number	Matrix A has the shape of rectangle
$\mathcal{R}$ is a set of real numbers	Has m rows and n columns
$\vec{v}$ has a length of n and contain real numbers $\vec{v} \in \mathcal{R}^n$	Use capital letter $A \in \mathcal{R}^{m \times n}$
$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \in \begin{bmatrix} \mathcal{R} \\ \mathcal{R} \\ \mathcal{R} \end{bmatrix} = \mathcal{R}^3$	$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \in \begin{bmatrix} \mathcal{R} & \mathcal{R} \\ \mathcal{R} & \mathcal{R} \\ \mathcal{R} & \mathcal{R} \end{bmatrix} = \mathcal{R}^{3 \times 2}$

Vector Addition

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_3 \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_3 \end{bmatrix}$$

$$\vec{v} + \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_3 \end{bmatrix} + \begin{bmatrix} u_1 \\ \dots \\ u_3 \end{bmatrix} = \begin{bmatrix} v_1 + u_1 \\ \dots \\ v_3 + u_3 \end{bmatrix}$$




```python
def add_vectors(vector1, vector2):
    '''
    Add corresponding elements between two vectors
    vector1 and vector2 are with list type
    output is a vector (list)
    '''

    return [v1+v2 for v1, v2 in zip(vector1, vector2)]
```

```python
import numpy as np

x = np.array([1,2,3,4])
y = np.array([5,6,7,8])

print('data x \n', x)
print('data y \n', y)

# Tổng của 2 mảng
print('method 1 \n', x + y)
print('method 2 \n', np.add(x, y))
```

```
data x
 [1 2 3 4]
data y
 [5 6 7 8]
method 1
 [ 6  8 10 12]
method 2
 [ 6  8 10 12]
```

Vector Operations

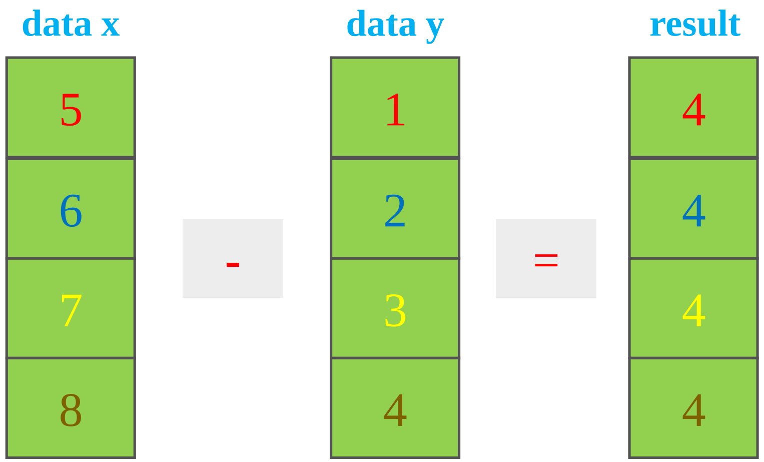
Vector subtraction

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} + \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} - \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 - u_1 \\ \dots \\ v_n - u_n \end{bmatrix}$$

```python
import numpy as np

x = np.array([5,6,7,8])
y = np.array([1,2,3,4])

print('data x \n', x)
print('data y \n', y)

# Hiệu 2 mảng
print('method 1 \n', x - y)
print('method 2 \n', np.subtract(x, y))
```

```
data x
 [5 6 7 8]
data y
 [1 2 3 4]
method 1
 [4 4 4 4]
method 2
 [4 4 4 4]
```



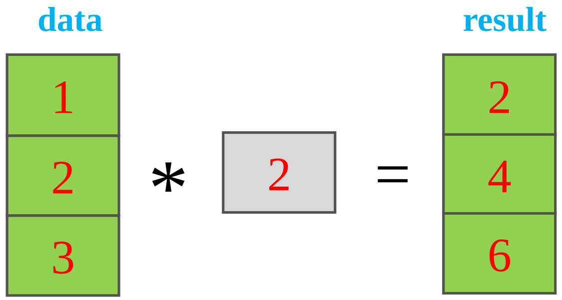
Vector Operations

Multiply with a number

$$\alpha\vec{u} = \alpha \begin{bmatrix} u_1 \\ \cdots \\ u_n \end{bmatrix} = \begin{bmatrix} \alpha u_1 \\ \cdots \\ \alpha u_n \end{bmatrix}$$

```python
import numpy as np

# create data
data = np.array([1,2,3])
factor = 2

# broadcasting
result_multiplication = data*factor
```

```
[1 2 3]
[2 4 6]
```



Length of a vector

$$\|\vec{u}\| = \sqrt{u_1^2 + \cdots + u_n^2}$$

```python
# compute length of a vector

import numpy as np

data = np.array([1, 2, 4, 2])
length = np.linalg.norm(data)
print(length)
```

```
5.0
```

Matrix Operations

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \dots & \dots & \dots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}$$

Addition

$$A + B = \begin{bmatrix} (a_{11} + b_{11}) & \dots & (a_{1n} + b_{1n}) \\ \dots & \dots & \dots \\ (a_{m1} + b_{m1}) & \dots & (a_{mn} + b_{mn}) \end{bmatrix}$$

Subtraction

$$A - B = \begin{bmatrix} (a_{11} - b_{11}) & \dots & (a_{1n} - b_{1n}) \\ \dots & \dots & \dots \\ (a_{m1} - b_{m1}) & \dots & (a_{mn} - b_{mn}) \end{bmatrix}$$

```python
import numpy as np

A = np.array([[4,2],
              [9,8]])
B = np.array([[1,2],
              [3,4]])

print(A+B)
print(A-B)
```

```
[[ 5  4]
 [12 12]]
[[3 0]
 [6 4]]
```

$$\underset{A}{\begin{bmatrix} 4 & 2 \\ 9 & 8 \end{bmatrix}} + \underset{B}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 5 & 4 \\ 12 & 12 \end{bmatrix}}$$

$$\underset{A}{\begin{bmatrix} 4 & 2 \\ 9 & 8 \end{bmatrix}} - \underset{B}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 3 & 0 \\ 6 & 4 \end{bmatrix}}$$

Vector Operations

Dot product

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} \cdot \vec{u} = v_1 \times u_1 + \cdots + v_n \times u_n$$

$$\underset{\textbf{v}}{\begin{bmatrix} 1 & 2 \end{bmatrix}} \bullet \underset{\textbf{w}}{\begin{bmatrix} 2 \\ 3 \end{bmatrix}} = \underset{\textbf{result}}{\begin{bmatrix} 8 \end{bmatrix}}$$

```python
def dot_product(vector1, vector2):
    '''
    Compute dot product between two vectors
    Output is a floating-point number
    '''

    return sum([v1*v2 for v1, v2 in zip(vector1, vector2)])

# test case
vector1 = [1, 2, 3]
vector2 = [2, 3, 4]

ouptut = dot_product(vector1, vector2)
print(ouptut)
```

```
20
```

```python
import numpy as np

v = np.array([1, 2])
w = np.array([2, 3])

# Tính inner product giữa v và w
print('method 1 \n', v.dot(w))
print('method 2 \n', np.dot(v, w))
```

```
method 1
8

method 2
8
```

Matrix Operations

Matrix-vector multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \qquad \vec{x} = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}$$

$A \in \mathcal{R}^{m \times n}$

$$C = A\vec{x} \quad \text{where } c_i = \sum_{l=1}^{n} a_{il} x_l$$

```python
# aivietnam.ai
import numpy as np

X = np.array([[1,2],
              [3,4]])
v = np.array([1,2])

print('matrix X \n', X)
print('vector v \n', v)

# phép nhân giữa ma trận và vector
print('method 1: X.dot(v)  \n', X.dot(v))
print('method 1: v.dot(X)  \n', v.dot(X))
#print('\n method 2: X.dot(v)  \n', np.dot(X, v))
#print('\n method 2: v.dot(X)  \n', np.dot(v, X))
```

```
matrix X
 [[1 2]
 [3 4]]
vector v
 [1 2]
method 1: X.dot(v)
 [ 5 11]
method 1: v.dot(X)
 [ 7 10]
```

$$\underset{X}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} \bullet \underset{v}{\begin{bmatrix} 1 \\ 2 \end{bmatrix}} = \underset{result}{\begin{bmatrix} 5 \\ 11 \end{bmatrix}}$$

$$\underset{v}{\begin{bmatrix} 1 & 2 \end{bmatrix}} \bullet \underset{X}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} = \underset{result}{\begin{bmatrix} 7 & 10 \end{bmatrix}}$$

Matrix Operations

Matrix-matrix multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}$$

$$A \in \mathcal{R}^{m \times n} \quad B \in \mathcal{R}^{n \times k}$$

$$C = AB$$

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}$$

$$AB = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix} = \begin{bmatrix} (a_{11}b_{11} + \cdots + a_{1n}b_{n1}) \dots (a_{1n}b_{1k} + \cdots + a_{1n}b_{nk}) \\ \dots \qquad \dots \qquad \dots \\ (a_{m1}b_{11} + \cdots + a_{mn}b_{n1}) \dots (a_{m1}b_{1k} + \cdots + a_{mn}b_{nk}) \end{bmatrix}$$

Matrix Operations

Matrix-matrix multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}$$

$$A \in \mathcal{R}^{m \times n} \qquad B \in \mathcal{R}^{n \times k}$$

$$C = AB$$

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}$$

```python
def matrix_multiplication(matrix1, matrix2):
    '''
    This function does the multiplication between two matrices.
    #columns of matrix1 == #rows of matrix2
    '''
    matrix1_nrows = len(matrix1)
    matrix1_ncols = len(matrix1[0])

    matrix2_nrows = len(matrix2)
    matrix2_ncols = len(matrix2[0])

    # tạo matrix kết quả
    result = [[0]*matrix2_ncols for i in range(matrix1_nrows)]

    for i in range(matrix1_nrows):
        for j in range(matrix2_ncols):
            for k in range(matrix2_nrows):
                result[i][j] += matrix1[i][k] * matrix2[k][j]

    return result

# test case
# 3x3 matrix
matrix1 = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]

# 3x4 matrix
matrix2 = [[1, 1, 2, 1],
           [1, 2, 1, 1],
           [1, 1, 1, 2]]

result = matrix_multiplication(matrix1, matrix2)
print(result[0])
print(result[1])
print(result[2])
```

```
[6, 8, 7, 9]
[15, 20, 19, 21]
[24, 32, 31, 33]
```

Matrix-matrix multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}$$

$$A \in \mathcal{R}^{m \times n} \qquad B \in \mathcal{R}^{n \times k}$$

$$C = AB$$

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}$$

```
# aivietnam.ai
import numpy as np

X = np.array([[1,2],
              [3,4]])
Y = np.array([[2,3],
              [2,1]])

# Phép nhân giữa hai ma trận
print('method 1 \n', X.dot(Y))
print('method 1 \n', Y.dot(X))
#print('method 2 \n', np.dot(X, Y))
#print('method 2 \n', np.dot(Y, X))
```

```
method 1
 [[ 6  5]
 [14 13]]
method 1
 [[11 16]
 [ 5  8]]
```

$$\overset{X}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} \bullet \overset{Y}{\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}} = \overset{\text{result}}{\begin{bmatrix} 6 & 5 \\ 14 & 13 \end{bmatrix}}$$

$$\overset{Y}{\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}} \bullet \overset{X}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} = \overset{\text{result}}{\begin{bmatrix} 11 & 16 \\ 5 & 8 \end{bmatrix}}$$

Hadamard product

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} \odot \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \odot \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 \times u_1 \\ \dots \\ v_n \times u_n \end{bmatrix}$$

data x		data y		result
1		5		5
2	*	6	=	12
3		7		21
4		8		32

```python
def Hadamard_product(vector1, vector2):
    '''
    Compute Hadamard product between two vectors
    Output is a vector
    '''
    return [v1*v2 for v1, v2 in zip(vector1, vector2)]
```

```python
import numpy as np

x = np.array([1,2,3,4])
y = np.array([5,6,7,8])

print('data x \n', x)
print('data y \n', y)

# Tích các phần tử tương ứng giữa x và y
print('method 1 \n', x*y)
print('method 2 \n', np.multiply(x, y))
```

```
data x
 [1 2 3 4]
data y
 [5 6 7 8]
method 1
 [ 5 12 21 32]
method 2
 [ 5 12 21 32]
```

Numpy Array Operations

❖ Division

data x		data y		result
1		5		0.2
2	/	6	=	0.33
3		7		0.42
4		8		0.5

```python
import numpy as np

x = np.array([1,2,3,4])
y = np.array([5,6,7,8])

print('data x \n', x)
print('data y \n', y)

# Phép chia các từng phần tương ứng x cho y
print('method 1 \n', x / y)
print('method 2 \n', x // y)
print('method 3 \n', np.divide(x, y))
```

```
data x
 [1 2 3 4]
data y
 [5 6 7 8]
method 1
 [0.2        0.33333333 0.42857143 0.5       ]
method 2
 [0 0 0 0]
method 3
 [0.2        0.33333333 0.42857143 0.5       ]
```

Matrix Operations

Transpose

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

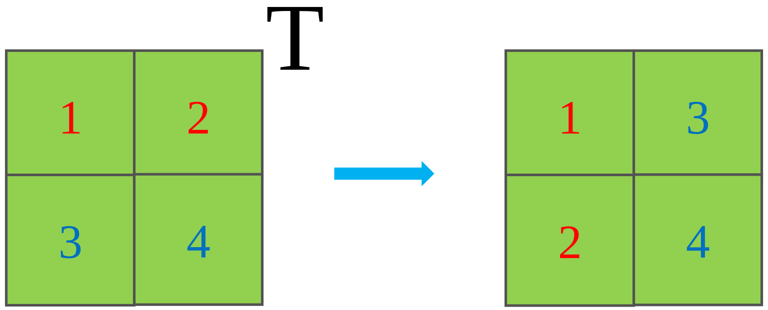
$$A^T = \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \dots & \dots & \dots \\ a_{1n} & \dots & a_{mn} \end{bmatrix}$$

```python
import numpy as np

X = np.array([[1,2],
              [3,4]])
print(X)

#chuyển vị
print(X.T)
```

```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
```

Example

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \quad A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \end{bmatrix}$$



Outline

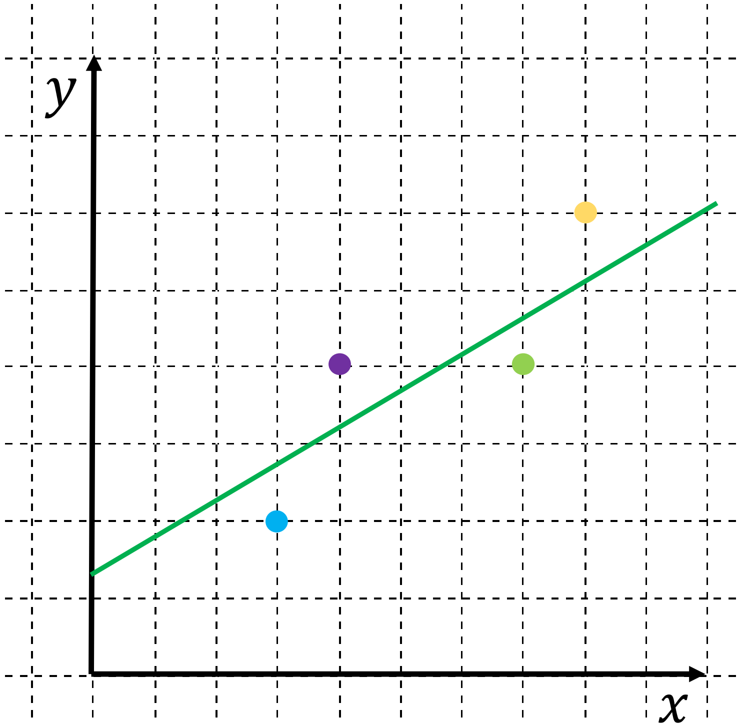
- Introduction to Numpy
- Numpy Examples
- Introduction to Vector and Matrix
- Vectorize the Linear Regression (1-sample)

Linear Regression

❖ Quick review

Linear regression models ← Linear equations

Linear equation = $w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$



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

House price data

Model: $\hat{y} = w_1 x_1 + b$

price = a $* \, area + b$

Linear Regression

❖ Quick review

Linear equation

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

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

w and b are parameters

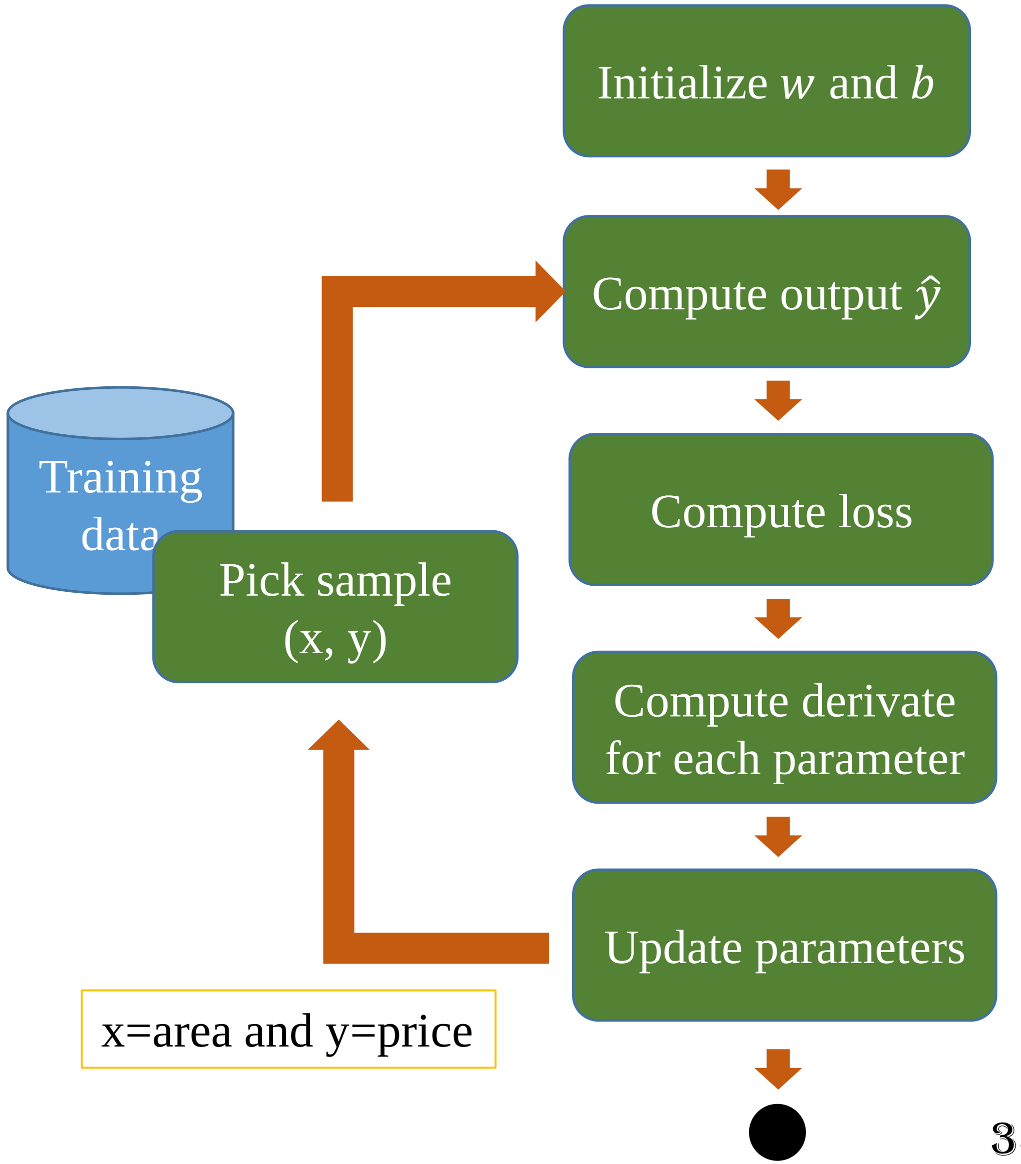
and x is input feature

Error (loss) computation

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

Squared loss

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



Linear Regression

❖ Quick review

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

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

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

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

η is learning rate

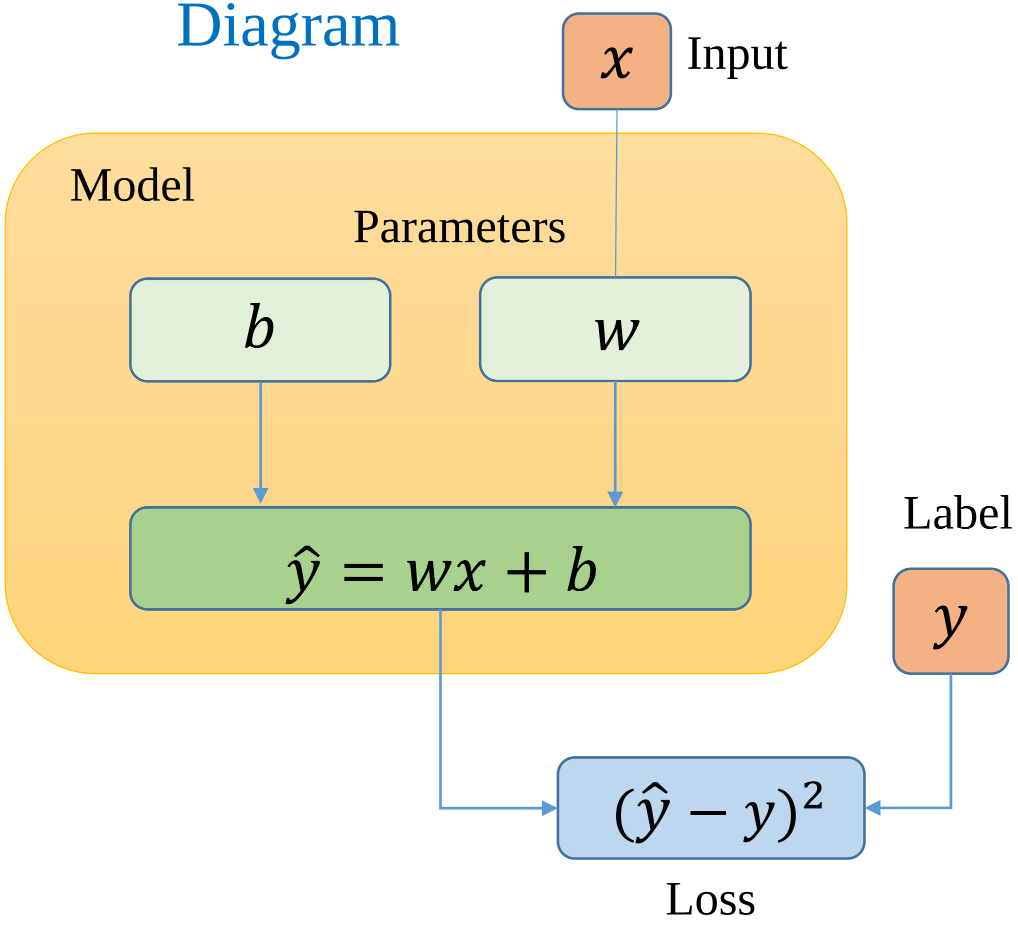
Linear Regression

❖ Quick review

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

House price data

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

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

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

Linear Regression

❖ Quick review

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

2) Tính output $\hat{y}$

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

3) Tính loss

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

4) Tính đạo hàm

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

5) Cập nhật tham số

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

```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

❖ Quick review

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

2) Tính output $\hat{y}$

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

3) Tính loss

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

4) Tính đạo hàm

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

5) Cập nhật tham số

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

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

# how long
epoch_max = 10
data_size = 4

for epoch in range(epoch_max):
    for i in range(data_size):
        # get a sample
        x = areas[i]
        y = prices[i]

        # predict y_hat
        y_hat = predict(x, w, b)

        # compute loss
        loss = (y_hat-y)*(y_hat-y)

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

        # update weights
        (w, b) = update_weight(w, b, lr, dw, db)
```

Problem and solution?

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

House price data

price = a $* area + b$

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: $\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + b$

Sale = $w_1 * TV + w_2 * Radio + w_3 * Newspaper + b$

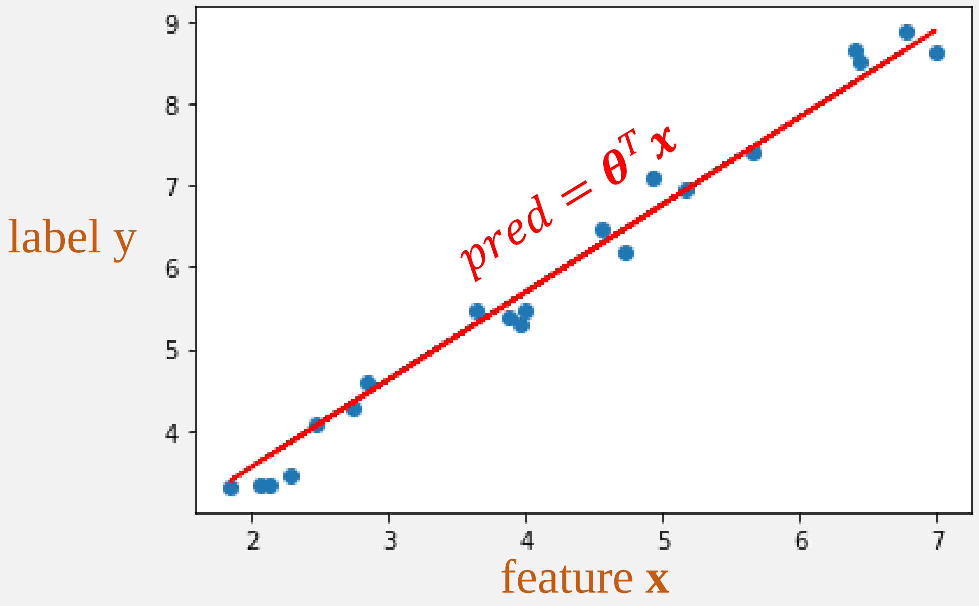
Features													Label
crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	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

medv = $w_1 * x_1 + \cdots + w_{13} * x_{13} + b$

Linear Regression

Model the relationship between feature x and label y



Using a linear equation to fit data

Samples (x, y) are given in advance

Linear equation

$$\hat{y} = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

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

$\boldsymbol{\theta} = [b \;\; w_1 \;\; w_2 \;...\; w_n]^T$ is parameter vector

and $\boldsymbol{x} = [1 \;\; x_1 \;\; x_2 \;...\; x_n]^T$ is feature vector.

Error (loss) computation

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

Squared loss

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

Linear Regression

Linear equation

$$\hat{y} = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

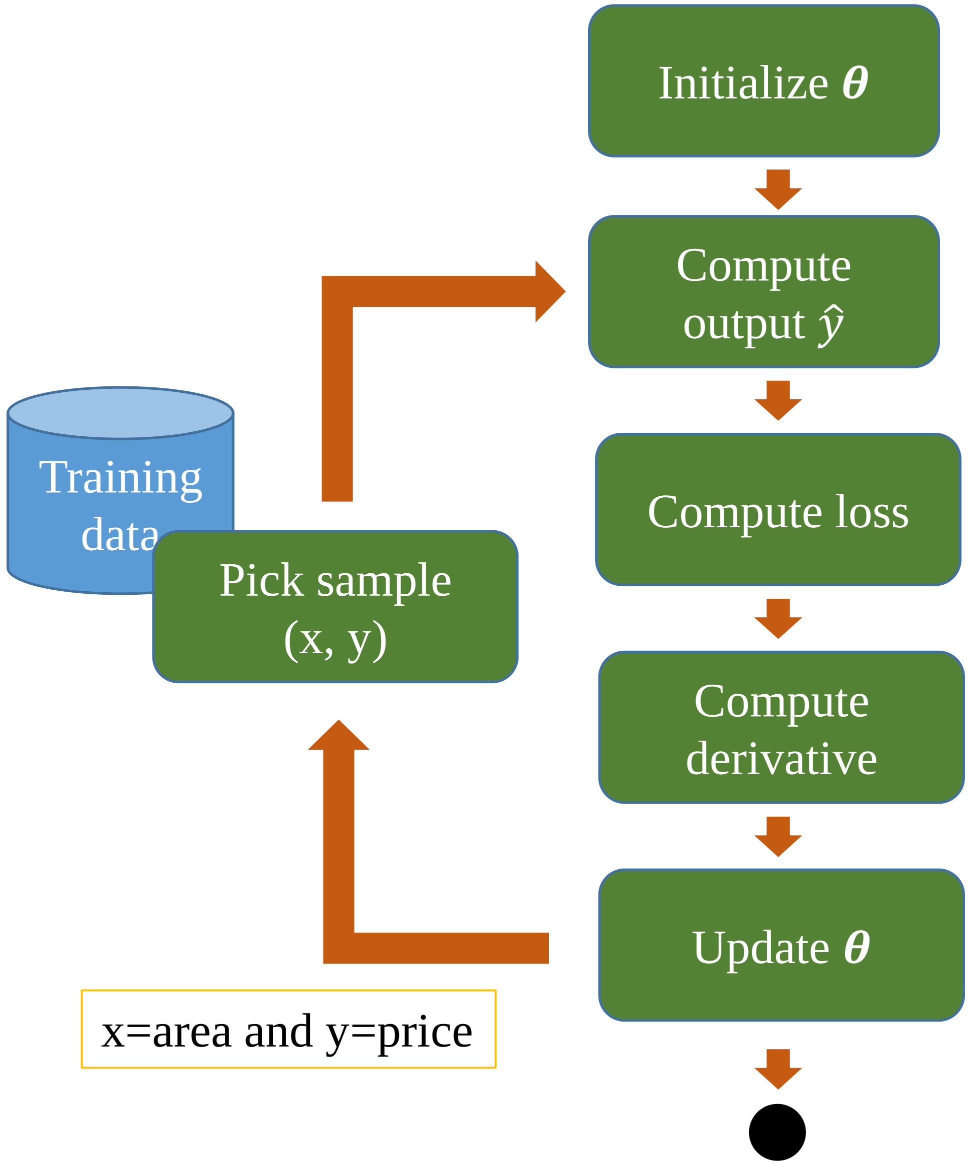
where $\hat{y}$ is a predicted value,
$\boldsymbol{\theta} = [b \;\; w_1 \;\; w_2 \; \ldots \; w_n]^T$ is parameter vector
and $\boldsymbol{x} = [1 \;\; x_1 \;\; x_2 \; \ldots \; x_n]^T$ is feature vector.

Error (loss) computation

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

Squared loss

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



Linear Regression: Vectorization

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

2) Compute 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

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

Normal version

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

2) Compute output $\hat{y}$

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{\theta}$$

3) Compute loss

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

4) Compute derivative

$$L'_{\boldsymbol{\theta}} = 2\boldsymbol{x}(\hat{y} - y)$$

5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta L'_{\boldsymbol{\theta}}$$

η is learning rate

Vectorization

Linear Regression: Vectorization

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

2) Compute output $\hat{y}$

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{\theta}$$

3) Compute loss

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

4) Compute derivative

$$L'_{\boldsymbol{\theta}} = 2\boldsymbol{x}(\hat{y} - y)$$

5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta L'_{\boldsymbol{\theta}}$$

η is learning rate

Vectorization

$$\underset{\mathbf{v}}{\begin{bmatrix} 1 & 2 \end{bmatrix}} \bullet \underset{\mathbf{w}}{\begin{bmatrix} 2 \\ 3 \end{bmatrix}} = \underset{\text{result}}{\begin{bmatrix} 8 \end{bmatrix}}$$

```python
import numpy as np

v = np.array([1, 2])
w = np.array([2, 3])

# Tính inner product giữa v và w
print('method 1 \n', v.dot(w))
print('method 2 \n', np.dot(v, w))
```

```
method 1
8

method 2
8
```

Linear Regression: Vectorization

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

2) Compute output $\hat{y}$

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{\theta}$$

3) Compute loss

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

4) Compute derivative

$$L'_{\boldsymbol{\theta}} = 2\boldsymbol{x}(\hat{y} - y)$$

5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta L'_{\boldsymbol{\theta}}$$

η is learning rate

Vectorization

```python
import numpy as np

# forward
def predict(x, theta):
    return x.dot(theta)

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

    return dtheta

# update weights
def update_weight(theta, lr, dtheta):
    dtheta_new = theta - lr*dtheta

    return dtheta_new
```

AI VIETNAM
All-in-One Course

Linear Regression Vectorization

❖ Implementation

θ =?

```python
import numpy as np

# forward
def predict(x, theta):
    return x.dot(theta)

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

    return dtheta

# update weights
def update_weight(theta, lr, dtheta):
    dtheta_new = theta - lr*dtheta

    return dtheta_new
```

```python
# test sample
x = np.array([6.7, 1])
y = np.array([9.1])

# init weight
lr = 0.01
theta = np.array([-0.34, 0.04]) #[w, b]
print('theta', theta)

# predict y_hat
y_hat = predict(x, theta)
print('y_hat: ', y_hat)

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

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

# update weights
theta = update_weight(theta, lr, dtheta)
print('theta_new: ', theta)
```