

# Python Tutorial

Sep. 28

# Outline

- Python
- Basic of Machine Learning
- TensorFlow
- Pytorch

# Python

- Introduction
- Installation and package managers
- Basic grammar (include OOPs)
- Tutorials of Useful Package

# Tools



- Life is short and you should use Python
- Python is a kind of programming languages that particularly suitable for machine learning researchers.
- It makes thing ridiculously simple by providing a huge amount of powerful packages.

<https://www.python.org/>

# Power of Python

- Suppose we have matrix A and B, and we want their product AB

## C++ example

```
C = new matrix[A.row_count, B.col_count]

for(int i=0; i < A.row_count, i++)
{
    for(int j=0; j<B.col_count; j++)
    {
        float s = 0;
        for(int k=0; k<A.col_count; k++)
        {
            s += A[i,k] *B[k, j];
        }
        C[i, j] = s;
    }
}
```

## Python example

```
C = A * B;
```

# Basics in Python

- Python is a kind of Interpreted language, instead of the compiled language like C++.
- It means that you can execute the code in the command line:

```
>>> print("Hello, World!")  
Hello, World!
```

- Or execute create a python file (.py extension):

```
C:\Users\Your Name>python myfile.py
```

- The detailed python tutorial can be found at <https://www.w3schools.com/python/default.asp>

# numpy



- Numpy is the most important Python library
- It is for large scale, high-dimensional data array processing and provides a large collection of high-level mathematical functions to operate on these arrays.
- Using NumPy, a developer can perform the following operations
  - Mathematical and logical operations on arrays.
  - Fourier transforms and routines for shape manipulation.
  - Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

```
>>> a = np.arange(6)                                # 1d array
>>> print(a)
[0 1 2 3 4 5]
>>>
>>> b = np.arange(12).reshape(4, 3)                 # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>>
>>> c = np.arange(24).reshape(2, 3, 4)             # 3d array
>>> print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```



# scipy



- Scipy is for scientific computing and technical computing.
- SciPy contains modules that are very common in science and engineering, such as
  - optimization,
  - linear algebra,
  - Integration
  - FFT,
  - signal and image processing
  - ODE solvers

$$\begin{aligned} 1x + 5y &= 6 \\ 3x + 7y &= 9 \end{aligned}$$

```
In [20]: from scipy import linalg

equation = np.array([[1, 5], [3, 7]])
solution = np.array([[6], [9]])

roots = linalg.solve(equation, solution)

print("Found the roots:")
print(roots)

print("\n Dot product should be zero if the solutions are correct:")
print(equation.dot(roots) - solution)
```

Found the roots:

```
[[0.375]
 [1.125]]
```

Dot product should be zero if the solutions are correct:

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

# scikit\_learn



- Scikit\_learn is a Python library for various classification, regression and clustering algorithms, including support vector machines, random forests, neural network, and so on.
- It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

# TensorFlow and PyTorch

- TensorFlow is the deep learning library developed by Google and PyTorch is the deep learning library developed by Facebook.
- They are the mostly used libraries by researchers all over the world.
- They are so important that we will have specific tutorials to teach them.

# Get started with Python

- Installation of Python
- Basics and Data Structure
- Useful Packages
- Datasets
- Solve a Machine Learning Problem

# Installation of Python

- For Windows
  - Download the latest [Python 3.x](#) version

- For Linux

- Most of the newer Linux based Operating system have already installed Python

- Check installation: `$ python3 --version`

- If not installed, then: `$ sudo apt-get update`  
`$ sudo apt-get install python3.6`

- For MacOS

- Most of the newer versions of MacOS have

- Check installation: `python --version`

- If not installed, then download the [Mac OS X 64-bit/32-bit](#) installer

## Python Releases for Windows

- [Latest Python 3 Release - Python 3.6.4](#)
- [Latest Python 2 Release - Python 2.7.14](#)
- [Python 3.6.4 - 2017-12-19](#)
  - [Download Windows x86 web-based installer](#)
  - [Download Windows x86 executable installer](#)
  - [Download Windows x86 embeddable zip file](#)
  - [Download Windows x86-64 web-based installer](#)
  - [Download Windows x86-64 executable installer](#)
  - [Download Windows x86-64 embeddable zip file](#)
  - [Download Windows help file](#)

## Python Releases for Mac OS X

- [Latest Python 3 Release - Python 3.6.4](#)
- [Latest Python 2 Release - Python 2.7.14](#)
- [Python 3.7.0a4 - 2018-01-09](#)
  - [Download Mac OS X 64-bit/32-bit installer](#)
- [Python 3.6.4 - 2017-12-19](#)
  - [Download Mac OS X 64-bit/32-bit installer](#)

# Python 2 vs. Python 3

- Python 2.x is legacy. Python 3.x is the present and future of the language  
Python 3.0 was released in 2008. The final 2.x version 2.7 release came out in mid-2010, with a statement of extended support for this end-of-life release.
- The difference between Python 2 and 3 is that Python 2 will get minimum support in future and Python 3 will continue to develop further in future.
- Both shares similar capabilities but some of their syntax are different.  
Whatever the version is both are used for building applications.
- Eg. Python 3: / means Floating Division; // means integer Division
- Python 2: float / float means Floating Division, like, 1.0 / 2
- int / int, int // float, float // float, float // int are float Div
- Eg. Print “Hello world” is only support in Python 2

# Package Managers

- Surely, after the installation of Python, you can start programming, We can't go through the details of a complex operation step by step. We can use the extension packages written by others
- Instead of going to the Home page of each package, we recommend Package Managers
  - Pip
  - Anaconda
  - Miniconda





# Anaconda

- Anaconda is a Python distribution that is particularly popular for data analysis and scientific computing
  - Includes Spyder, a Python development environment
  - Includes conda, a platform-independent package manager
- Installing Python packages with Conda
  - ```
$ conda install tensorflow-gpu
```
  - Specific versions of packages can be requested:  

```
$ conda install tensorflow-gpu=1.15.0
```
  - Some packages are only available in special channels:  

```
$ conda install -c vpython vpython
```

# Anaconda

- Create virtual environments
  - Environments allow different versions of packages on same machine
  - Create environment “test” and install numpy version 1.7:

```
$ conda create --name test numpy=1.7
```

- Environments must be activated

- For MacOS and Linux

```
$ source activate test  
$ python  
...  
$ source deactivate
```

- For Windows

```
> activate test  
> python  
...  
> deactivate
```

# Download from mirror to increase speed

- Modify the ``.condarc`` under the user folder, with

channels:

- defaults

show\_channel\_urls: true

channel\_alias: <https://mirrors.tuna.tsinghua.edu.cn/anaconda>

default\_channels:

- <https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/main>
- <https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/free>
- <https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/r>
- <https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/pro>
- <https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/msys2>

custom\_channels:

conda-forge: <https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud>

msys2: <https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud>

bioconda: <https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud>

menpo: <https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud>

pytorch: <https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud>

simpleitk: <https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud>

# Anaconda and Miniconda

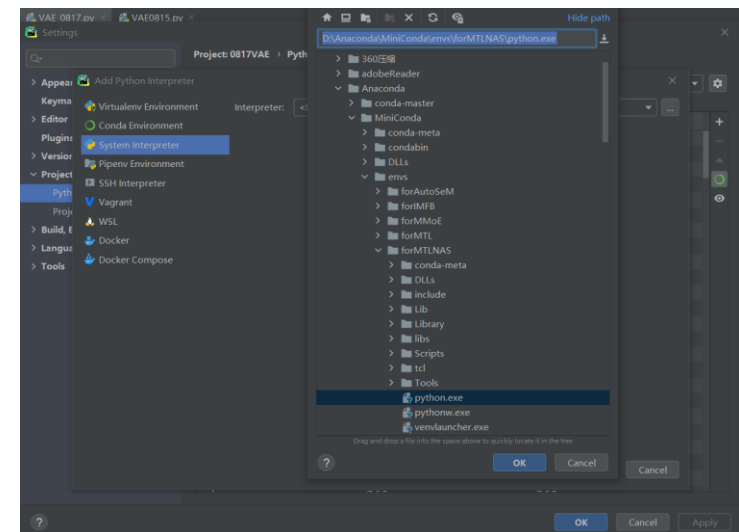
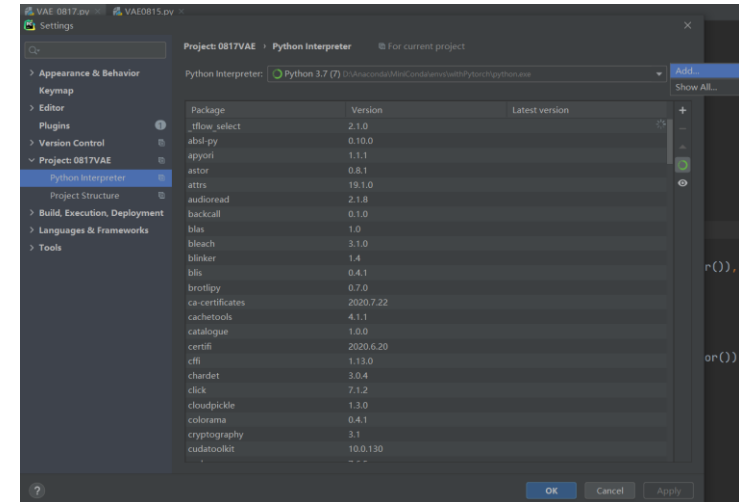
- Miniconda is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others.
- Choose Anaconda if you:
  - Are new to conda or Python
  - Like the convenience of having Python and over 150 scientific packages automatically installed at once
  - Have the time and disk space (a few minutes and 3 GB), and/or
  - Don't want to install each of the packages you want to use individually.
- Choose Miniconda if you:
  - Do not mind installing each of the packages you want to use individually.
  - Do not have time or disk space to install over 150 packages at once, and/or
  - Just want fast access to Python and the conda commands, and wish to sort out the other programs later.

# Python Basics and Data Structure

- Main Function
- Variables
- IF/ELSE
- Loops
- OOPs
- Functions
- Arrays
- Modules

# Create your First Python Program

- IDE: Spyder, PyCharm, etc.
- Create .py file
- Type a simple program
- Run the program
- Check the output



# Main Function

- Consider the following code

```
def main():  
    print("Hello world!")  
  
print("AI2020")
```

- Run it, what is the output?

```
D:\Anaconda\MiniConda\envs\forRRS2\python.exe "G:/Python  
workspace/AI2020/tutorial.py"  
AI2020
```

- Why? It is because we did not declare the call **function** `if __name__ == "__main__"`.



# Main Function

- When Python interpreter reads a source file, it will execute all the code found in it.
- When Python runs the "source file" as the main program, it sets the special variable (`__name__`) to have a value ("`__main__`").
- When you execute the main function, it will then read the "if" statement and checks whether `__name__` does equal to `__main__`.
- In Python, `if __name__ == "__main__"` allows you to run the Python files either as `reusable modules` or `standalone programs`.

# Main Function

- Run the following code

```
def main():  
    print("Hello world!")  
if __name__ == "__main__":  
    main()  
print("AI2020")
```

- Output

```
D:\Anaconda\MiniConda\envs\forRRS2\python.exe "G:/Python  
workspace/AI2020/tutorial.py"  
Hello world!  
AI2020
```

# Variables

- Declare and use a Variable

```
# Declare a variable and initialize it
f = 0
print(f)
# Re-declaring the variable
f = 'AI2020'
print(f)
```

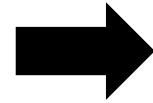
- What is the output?

```
D:\Anaconda\MiniConda\envs\forRRS2\python.exe "G:/Python
workspace/AI2020/tutorial.py"
0
AI2020
```

# Variables

- Concatenate Variables

```
a = "AI"  
b = 2020  
print(a + b)
```



```
a = "AI"  
b = 2020  
print(a + str(b))
```

- What is the output?

```
D:\Anaconda\MiniConda\envs\forRRS2\python.exe "G:/Python  
workspace/AI2020/tutorial.py"  
Traceback (most recent call last):  
  File "G:/Python workspace/AI2020/tutorial.py", line 19, in  
<module>  
    print(a + b)  
TypeError: can only concatenate str (not "int") to str
```

```
D:\Anaconda\MiniConda\envs  
\forRRS2\python.exe  
"G:/Python  
workspace/AI2020/tutorial  
.py"  
AI2020
```

# Variables

- Local & Global Variables

```
# Declare a variable and initialize it
a = 2020
print(a)
# Global vs. Local variables in functions
def someFunction():
    # local f
    a = "I am learning Python"
    print(a)
someFunction()
print(a)
```

- Output?

```
D:\Anaconda\MiniConda\envs
\forRRS2\python.exe
"G:/Python
workspace/AI2020/tutorial
.py"
2020
I am learning Python
2020
```

# Functions

- Define and call a function in Python

```
def square(x):  
    return x*x  
  
print(square(4))
```

- Any args or input parameters should be placed within these parentheses
- The code within every function starts with a colon (:) and should be **indented** (space)
- return means exits this function and pass back a value to the caller. A return without args means “return None”.

# IF/ELSE

- if, elif, else
- switch

```
def SwitchExample(argument):  
    switcher_with_map = {  
        0: "This is Case Zero",  
        1: "This is Case One",  
        2: "This is Case Two"  
    }  
    return switcher_with_map.get(argument, "default")  
  
if __name__ == "__main__":  
    argument = 1  
    print(SwitchExample(argument))
```

```
def main():  
    x, y = 20, 21  
    if x < y:  
        st = "x is less than y"  
    elif x == y:  
        st = "x is same as y"  
    else:  
        st = "x is greater than y"  
    print(st)  
  
if __name__ == "__main__":  
    main()
```

# Loops

- While loop

```
x = 0
# define a while loop
while x < 4:
    print(x)
    x = x + 1
```

```
0
1
2
3
```

- For loop

```
for x in range(2, 7):
    print(x)
```

```
2
3
4
5
6
```

- break and continue

```
for x in range(10, 20):
    if x == 19:
        break
    if x % 5 == 0:
        continue
    print(x)
```

```
11
12
13
14 ←
16
17
18 ←
```

```
Months = ["Jan", "Feb", "Mar",
           "April", "May", "June"]
for m in Months:
    print(m)
```

```
Jan
Feb
Mar
April
May
June
```



# OOPs: Class, Object, Inheritance and Constructor

- Define Python classes

```
class MyClass(object):  
    def method1(self):  
        print("AI2020")  
    def method2(self, input_string):  
        print("Class testing: " + input_string)
```

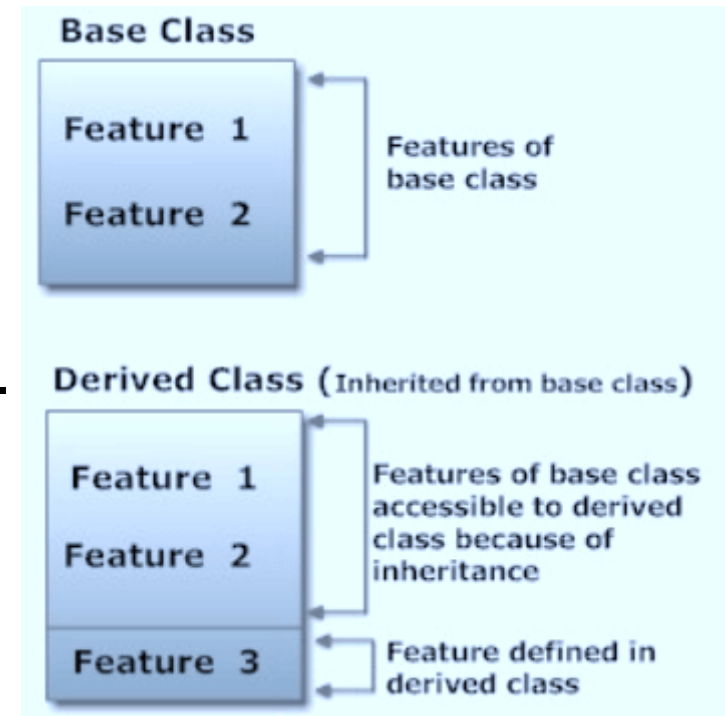
- To make an object of the class and call a method

```
c = MyClass()  
c.method1()  
c.method2("Hello Class")
```

```
AI2020  
Class testing: Hello Class
```

# Inheritance

- Inheritance refers to defining a new class with less or no modification to an existing class.
- The new class is called **derived class** and from one which it inherits is called the **base**.
- Python supports inheritance; it also supports **multiple inheritances**.
- `class DerivedClass(BaseClass):`  
    body\_of\_derived\_class



# Inheritance

- childClass inherits myClass

```
class ChildClass(MyClass):  
    def method2(self):  
        print("execute childClass method2")  
  
c2 = ChildClass()  
c2.method1()  
c2.method2()
```

- method1 is not defined but it is derived from the parent myClass.

```
AI2020 ← derived  
execute childClass method2 ← new
```

# Class

- The self-argument refers to the **object itself**. Hence the use of the word self. So inside this method, self will refer to **the specific instance of this object that's being operated on**.
- Self is the name preferred by convention by Python to indicate **the first parameter of instance methods** in Python. It is part of the Python syntax to access members of objects

# Constructors

- A constructor is a class function that instantiates an object to predefined values.
- It begins with a double underscore (`_`). It `__init__()` method

```
class User:
    def __init__(self, name):
        self.name = name

    def say_hello(self):
        print("Welcome to AI2020, " + self.name)

User1 = User("Alex")
User1.say_hello()
```

# Arrays

- Python has no native implementation for arrays but instead features **tuples**, **lists**, and **dictionaries**.
- List: simple list of values; index starts at 0

```
students = ['Sam', 'Tom', "Jack", "Dave", "Lucy"]
print(students[2])
students.append('Catherine')    # append to tail
print(students)
del students[3]                 # delete an element
print(students)
print(students[2:5])           # slice the list
```

```
Jack
['Sam', 'Tom', 'Jack', 'Dave',
 'Lucy', 'Catherine']
['Sam', 'Tom', 'Jack', 'Lucy',
 'Catherine']
['Jack', 'Lucy', 'Catherine']
```

- Tuples: similar to lists but **read-only**

```
# tuple
```

```
days = ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')
```

# Dictionaries

- Dictionaries: values are not numbered, neither are they ordered; values can be accessed using keys

```
gradebook = {"Jack": ["a-", "b", "c+"],  
             "Tom": ["a", "c", "a+"]}  
del gradebook["Jack"]  
gradebook["Catherine"] = ["b", "b+", "a"]  
print(gradebook)
```

```
{'Tom': ['a', 'c', 'a+'],  
 'Catherine': ['b', 'b+', 'a']}
```

# Modules

- A module is a “Python library”, that is a file where some functions, variables, or classes are defined.
- Modules can be easily installed using the package manager
  - Use Anaconda to install packages (or modules)
  - E.g. Open Anaconda prompt and type in command:  
conda install numpy  
...
- Import modules in .py file

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```



# Modules

- Write our own modules in .py files
- E.g. write a module in leo.py

```
tutorial.py × Add_file.py ×  
1 def add(a, b):  
2     res = a + b  
3     return res  
4  
5  
6 if __name__ == "__main__":  
7     print(add(3, 4))
```

```
D:\Anaconda\MiniConda\envs  
\forRRS2\python.exe  
"G:/Python  
workspace/AI2020/Add_file.py"  
7
```

use it in tutorial.py

```
tutorial.py × Add_file.py ×  
1 import Add_file  
2 print(Add_file.add(1, 2))  
3
```

```
D:\Anaconda\MiniConda\envs  
\forRRS2\python.exe  
"G:/Python  
workspace/AI2020/tutorial.py"  
3
```

# Tutorials of Useful Package

- Numpy
- Scipy
- Scikit-learn
- Matplotlib

# Numpy

- The NumPy (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data.
- Installation: `pip install numpy`
- Importation
  - Import and give a briefer name: `import numpy as np`, so we can further use `np.x`
  - Import directly into the current namespace: `from numpy import *`, so that we don't have to use dot notation at all

# Numpy

- The central feature of NumPy is the array object class.
- Every element of an array must be of the same type
- An array can be created from a list: `a = np.array([1, 4, 5, 8], float)`
- Array elements are accessed, sliced, and manipulated just like lists:
  - Run `a[:2]` , `a[3]` , `a[0] = 5`
- Use of a single ":" in a dimension indicates the use of everything along that dimension `a = np.array([[1, 2, 3], [4, 5, 6]], float)`
  - Run `a[0,1]` , `a[1,:]` , `a[-1:-2:]`
- Access the information of array
  - Run `a.shape` , `a.dtype` , `len(a)`

# Numpy

- Reshape.

```
a = np.array(range(6), float).reshape((2, 3))
a.transpose()
```

- Concatenate

```
a = np.array([1,2], float)
b = np.array([3,4,5,6], float)
c = np.array([7,8,9], float)
np.concatenate((a, b, c))
```

- Array with zeros and ones

```
np.ones((2,3), dtype=float)
np.zeros(7, dtype=int)
```

- Array mathematics

```
a = np.array([1,2,3], float)
b = np.array([5,2,6], float)
a + b
a - b
a * b
np.dot(b, a)
b / a
```

- Basic array operations

```
a = np.array([2, 4, 3], float)
a.sum()
a.prod()
a.mean()
a.var()
```

# SciPy

- SciPy greatly extends the functionality of the NumPy routines. We will not cover this module in detail but rather mention some of its capabilities.
- Access by simply importing the module: `import scipy`

| module          | code for...                                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| scipy.constants | Many mathematical and physical constants.                                                                                 |
| scipy.special   | Special functions for mathematical physics, such as iry, elliptic, bessel, gamma, beta, hypergeometric                    |
| scipy.integrate | Functions for performing numerical integration. Also provides methods for integration of ordinary differential equations. |
| scipy.optimize  | Standard minimization / maximization routines that operate on generic user-defined objective functions.                   |
| ...             |                                                                                                                           |

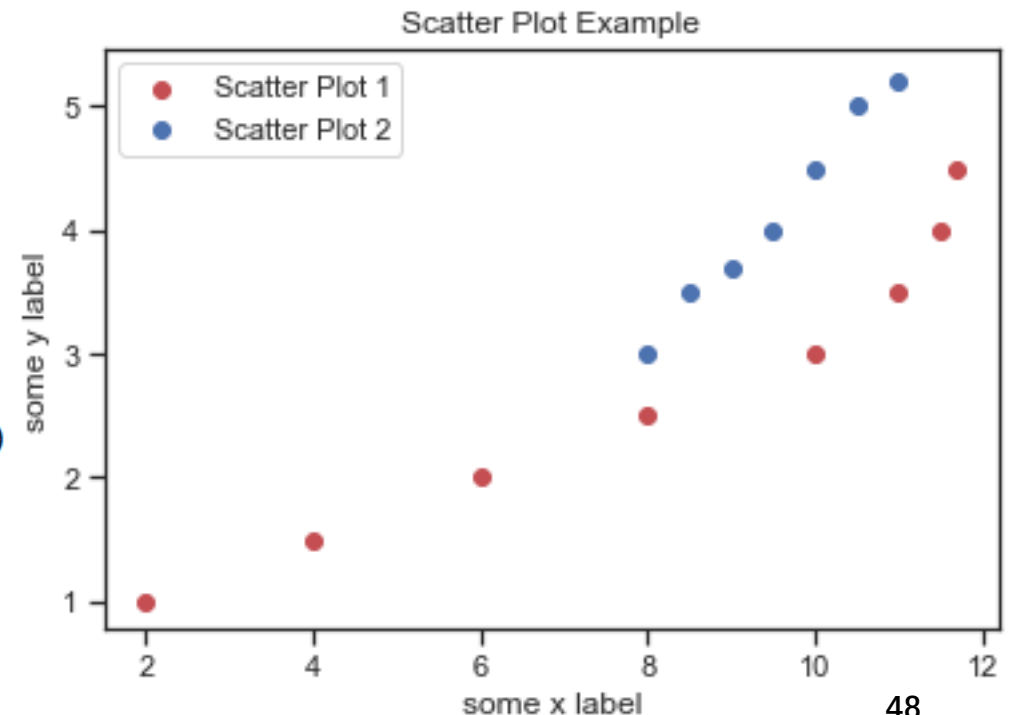
# Scikit-learn

- Scikit-learn is used to build the Machine Learning models
- Access by `from sklearn import svm` (E.g. import SVM model)
- Create a model `model1=svm.SVC(kernel='linear')`
- Train it with a training set `model1.fit(sample_train,label_train)`
- Use the trained model on a test set `prediction=model1.predict(sample_test)`
- Check the accuracy `print('the accuracy of the SVM (linear) is:',  
metrics.accuracy_score(prediction,label_test))`

# Matplotlib

- Matplotlib is a Python library used for plotting.
- Access by `import matplotlib.pyplot as plt`
- Scatter Plot

```
# data  
a = [2,4,6,8,10,11,11.5,11.7]  
b = [1,1.5,2,2.5,3,3.5,4,4.5]  
ab=[8,8.5,9,9.5,10,10.5,11]  
cd=[3,3.5,3.7,4,4.5,5,5.2]  
# matplotlib plot  
plt.scatter(a,b,label='Scatter Plot 1',color='r')  
plt.scatter(ab,cd,label='Scatter Plot 2',color='b')  
plt.xlabel('some x label')  
plt.ylabel('some y label')  
plt.title('Scatter Plot Example')  
plt.legend()  
plt.show()
```

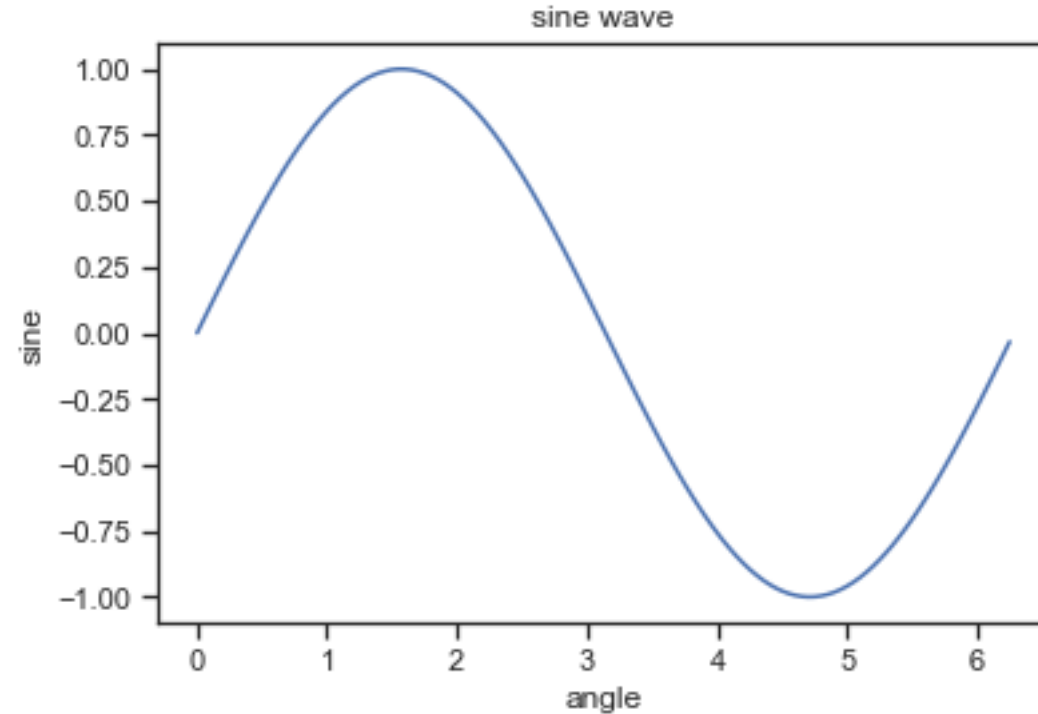




# Matplotlib

- Basic Plot

```
import math
x=np.arange(0, math.pi*2, 0.05)
y=np.sin(x)
plt.plot(x,y)
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
plt.show()
```



# Outline

- Python
- Basic of Machine Learning
- TensorFlow
- Pytorch

# Basic of Machine Learning

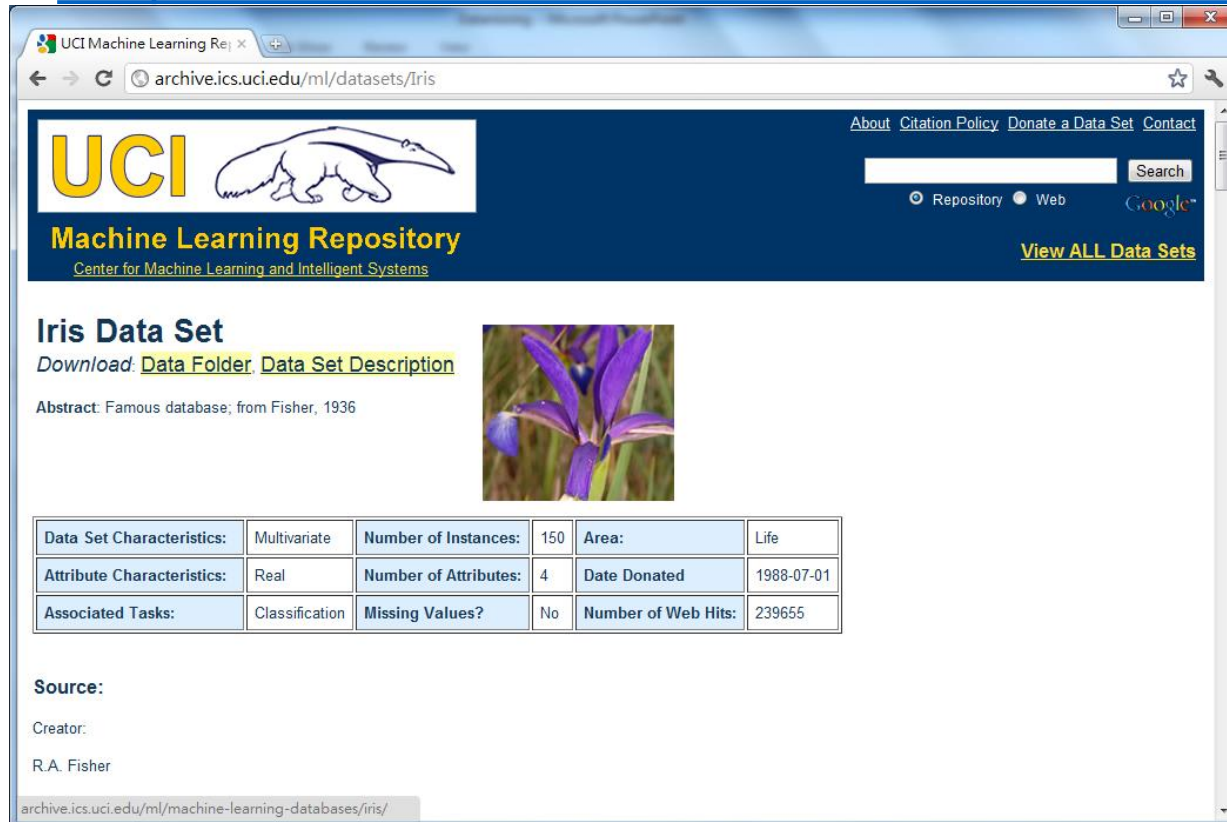
- Data
- Basic of Supervised Learning
- Gradient Descent

# Machine learning and data

- Everything is learned from data, and everything is evaluated on data.
- Machine learning is not only about modeling, it is also about data.
- In real applications, most time is spent on data preprocessing and cleaning.
- Some important public accessible dataset will be introduced.

# UCI machine learning dataset

- <https://archive.ics.uci.edu/ml/datasets.php>



The screenshot shows the UCI Machine Learning Repository website. The header includes the UCI logo, a search bar, and navigation links. The main content area is titled "Iris Data Set" and includes a small image of a purple iris flower. Below the image is a table with data set characteristics.

|                            |                |                       |     |                     |            |
|----------------------------|----------------|-----------------------|-----|---------------------|------------|
| Data Set Characteristics:  | Multivariate   | Number of Instances:  | 150 | Area:               | Life       |
| Attribute Characteristics: | Real           | Number of Attributes: | 4   | Date Donated        | 1988-07-01 |
| Associated Tasks:          | Classification | Missing Values?       | No  | Number of Web Hits: | 239655     |

Source:  
Creator:  
R.A. Fisher

The most basic and famous dataset for newbies of machine learning

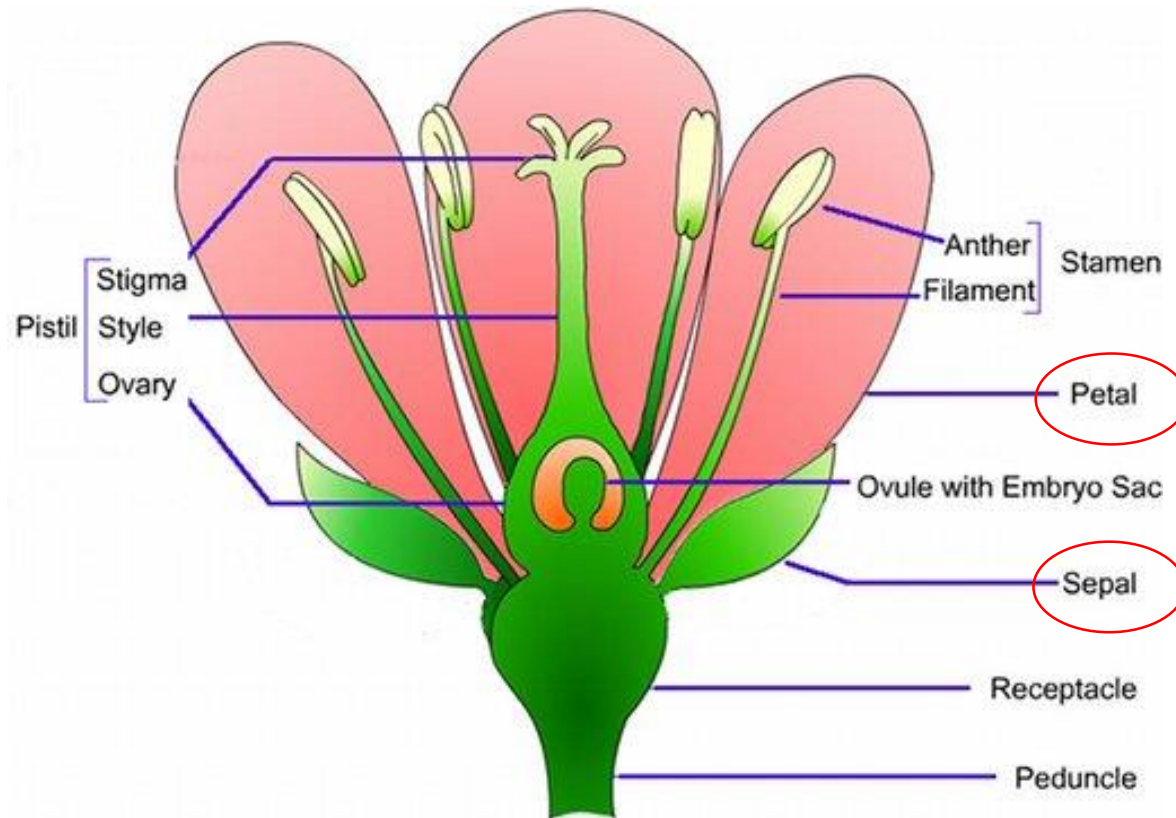
It's not suitable for research (too simple and too easy)

It's quite suitable for education

# Example of UCI dataset : Iris

- <https://archive.ics.uci.edu/ml/datasets/Iris>.

150 samples  
4 features  
3 classes.



## 7. Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

# MNIST dataset

- <http://yann.lecun.com/exdb/mnist/>
- Each digit is a  $28 \times 28 = 784$  grey image
- 60,000 images for training
- 10,000 images for testing
- 10 classes.



# ImageNet dataset

- <http://image-net.org/download>
- More than 14 million images
- About 20,000 classes
  
- The most important benchmark dataset in image processing
- DNN was first used in ImageNet challenge in 2012
- The accuracy of Computer outperformed that of human in 2015;





# NLP dataset

- Amazon Fine Food Reviews. This dataset consists of reviews of fine foods from amazon.
- Reviews include product and user information, ratings, and a plain text review.
  - Reviews from Oct 1999 - Oct 2012
  - 568,454 reviews
  - 256,059 users
  - 74,258 products
  - 260 users with > 50 reviews
- <https://www.kaggle.com/snap/amazon-fine-food-reviews>

# DrugBank

- <https://www.drugbank.ca/>
- 2,611 approved small molecule drugs,
- 1,300 approved biotech (protein/peptide) drugs,
- 130 nutraceuticals
- 6,315 experimental drugs
- Millions of interactions among these items.



# Basic of Supervised Learning

- Supervised Learning
- Progress
- Solution

# Supervised Learning

- Given: Training examples

$$\{(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_P, f(x_P))\}$$

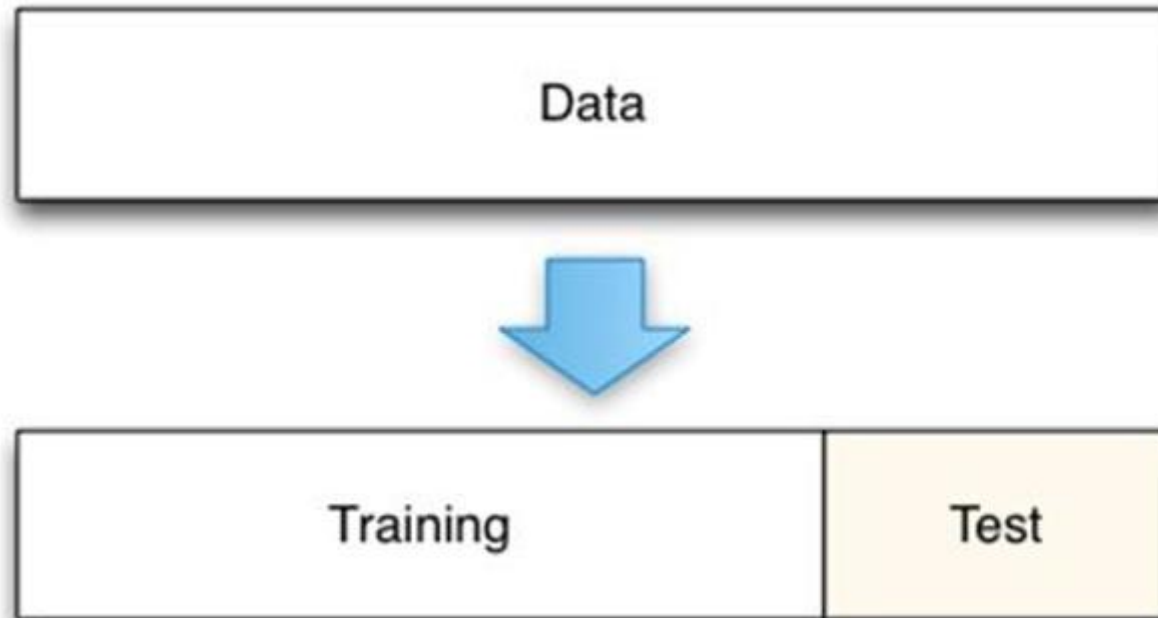
- of some unknown function (system)  $y = f(x)$
- Find  $\hat{f}(x)$  (i.e., an approximation)
- Predict  $y' = \hat{f}(x')$ , where  $x'$  is not in the training set

# Two Types of Supervised Learning

- Classification
  - Predict which category the input belongs to.
- Regression
  - Predict the numerical output corresponding to the input.
  - stock features to stock price

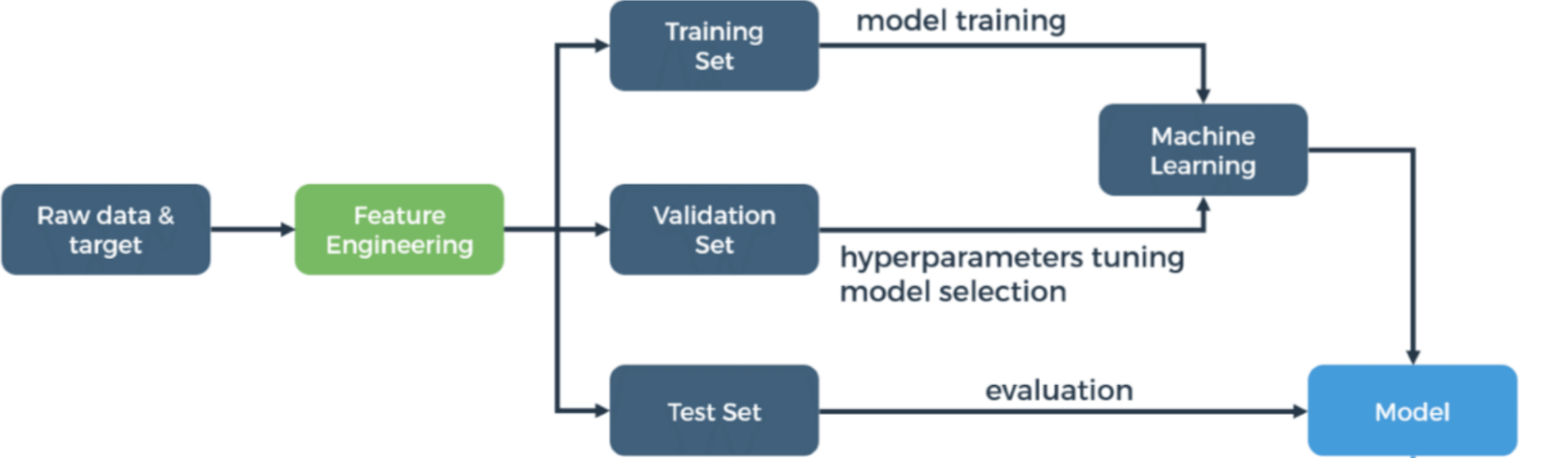
# Dataset

- Split dataset to training and test
- Training models on training dataset
- The evaluation of the model is the error on test dataset



# Process

## TRAINING

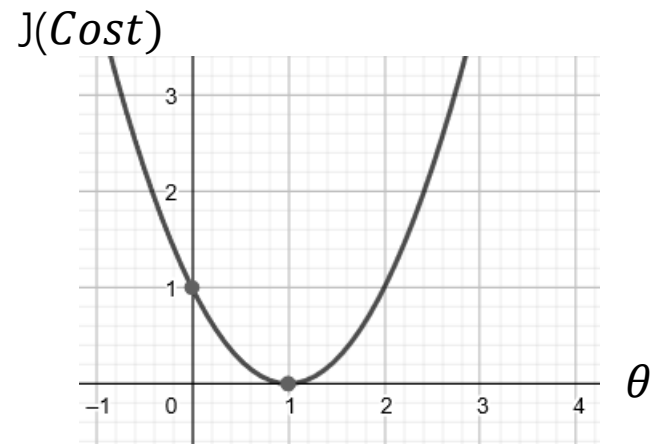


---

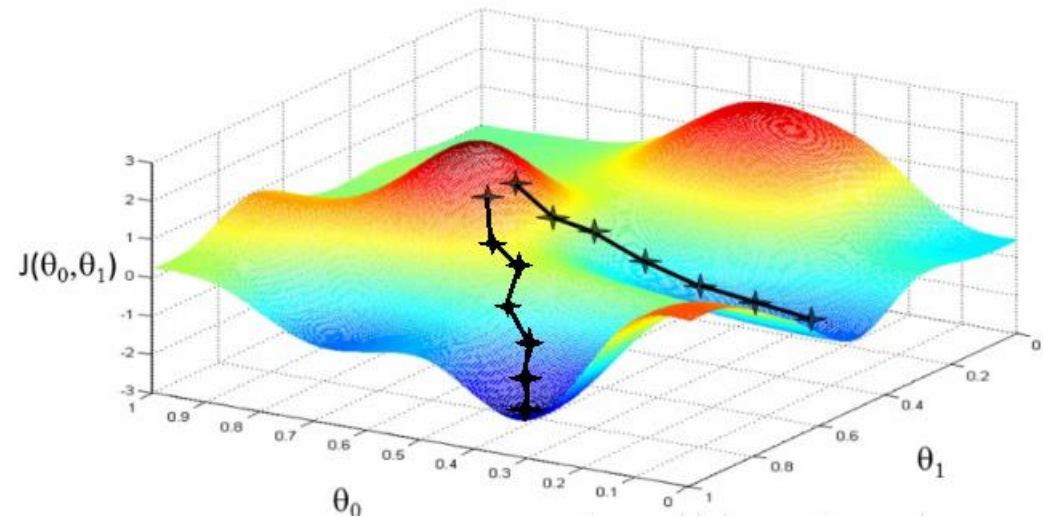
## PREDICTING



# Gradient Descent



$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J$$



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

- Numpy has Narray support, but doesn't offer methods to compute derivative (+ no GPU support)



# Outline

- Python
- Basic of Machine Learning
- TensorFlow
- Pytorch

# Tutorials of TensorFlow (1.15.0)

- Construct calculation Graph
- Calculation operation
- Example of Linear Regression

# TensorFlow Mechanics

- 2 feed data and run graph (operation)  
`sess.run (op, feed_dict={x: x_data})`

- 1 Build graph using TensorFlow operations



- 3 update variables in the graph (and return values)

# How to build a Calculation graph

- Build Graph

```
# Build the Calculation Graph
a = tf.constant(1.0)
b = tf.constant(2.0)
c = (b + a) * b
print(c)
```

```
Tensor("mul:0", shape=(),
      dtype=float32)
```

- Calculate in session

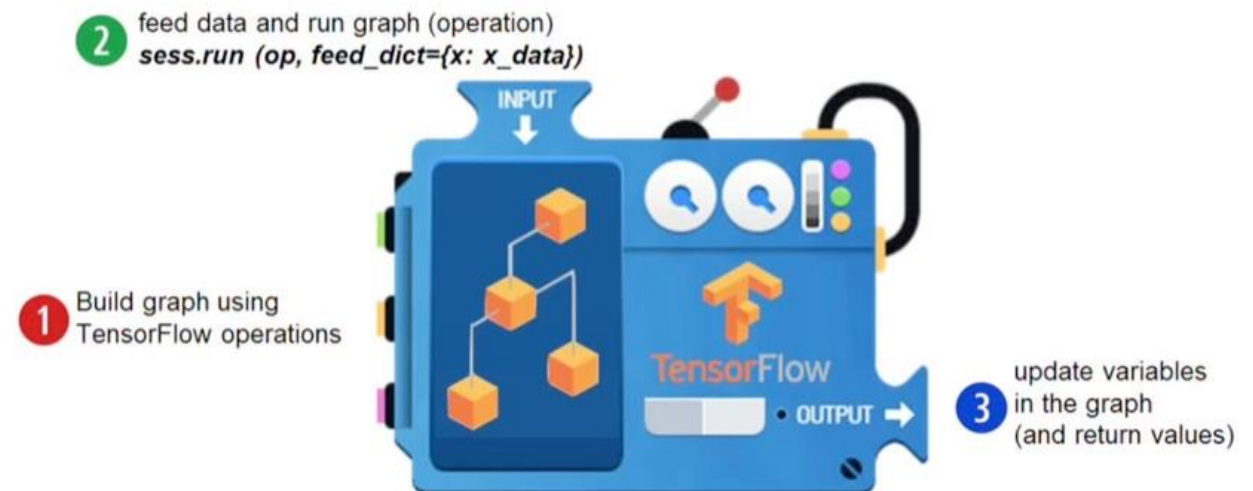
```
# Build the Calculation Graph
a = tf.constant(1.0)
b = tf.constant(2.0)
c = (b + a) * b
# A Session provide the environment where
# Tensor objects are evaluated
with tf.Session() as sess:
    print(sess.run(c))
    # Another way of writing
    print(c.eval())
```

```
6.0
6.0
```

# TensorFlow Mechanics

- TensorFlow programs are usually structured into a construction phase, that **assembles a graph**, and an execution phase that **uses a session to execute ops in the graph**. –TensorFlow docs

## TensorFlow Mechanics



# Calculation operation

| Numpy                                                | TensorFlow                                           |
|------------------------------------------------------|------------------------------------------------------|
| <code>a = np.zeros((2,2)); b = np.ones((2,2))</code> | <code>a = tf.zeros((2,2)), b = tf.ones((2,2))</code> |
| <code>np.sum(b, axis=1)</code>                       | <code>tf.reduce_sum(a, reduction_indices=[1])</code> |
| <code>a.shape</code>                                 | <code>a.get_shape()</code>                           |
| <code>np.reshape(a, (1,4))</code>                    | <code>tf.reshape(a, (1,4))</code>                    |
| <code>b * 5 + 1</code>                               | <code>b * 5 + 1</code>                               |
| <code>np.dot(a,b)</code>                             | <code>tf.matmul(a, b)</code>                         |
| <code>a[0,0], a[:,0], a[0,:]</code>                  | <code>a[0,0], a[:,0], a[0,:]</code>                  |

# Variable (Can be updated )

```
In [32]: W1 = tf.ones((2,2))
```

```
In [33]: W2 = tf.Variable(tf.zeros((2,2)), name="weights")
```

```
In [34]: with tf.Session() as sess:  
         print(sess.run(W1))  
         sess.run(tf.initialize_all_variables())  
         print(sess.run(W2))
```

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



*Note the initialization step `tf.initialize_all_variables()`*

# Updating Variable State

```
In [63]: state = tf.Variable(0, name="counter")
```

```
In [64]: new_value = tf.add(state, tf.constant(1)) ← Roughly new_value = state + 1
```

```
In [65]: update = tf.assign(state, new_value) ← Roughly state = new_value
```

```
In [66]: with tf.Session() as sess: Roughly  
.....:     sess.run(tf.initialize_all_variables()) state = 0  
.....:     print(sess.run(state)) print(state)  
.....:     for _ in range(3): ← for _ in range(3):  
.....:         sess.run(update) state = state + 1  
.....:         print(sess.run(state)) print(state)  
.....:
```

0  
1  
2  
3



# How can we input data into TensorFlow

- All previous examples have manually defined tensors. How can we input external data into TensorFlow?
- Simple solution: Import from Numpy

```
In [93]: a = np.zeros((3,3))
In [94]: ta = tf.convert_to_tensor(a)
In [95]: with tf.Session() as sess:
.....:     print(sess.run(ta))
.....:
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
```

# Placeholders and Feed Dictionaries

- Inputting data with `tf.convert_to_tensor()` is convenient, but doesn't scale.
- Use `tf.placeholder` variables (dummy nodes that provide entry points for data to computational graph).
- A `feed_dict` is a python dictionary mapping from `tf.placeholder` vars (or their names) to data (numpy arrays, lists, etc.).

# Placeholders and Feed Dictionaries

```
In [96]: input1 = tf.placeholder(tf.float32)
```

```
In [97]: input2 = tf.placeholder(tf.float32)
```

```
In [98]: output = tf.mul(input1, input2)
```

```
In [99]: with tf.Session() as sess:
```

```
.....:     print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))
```

```
.....:
```

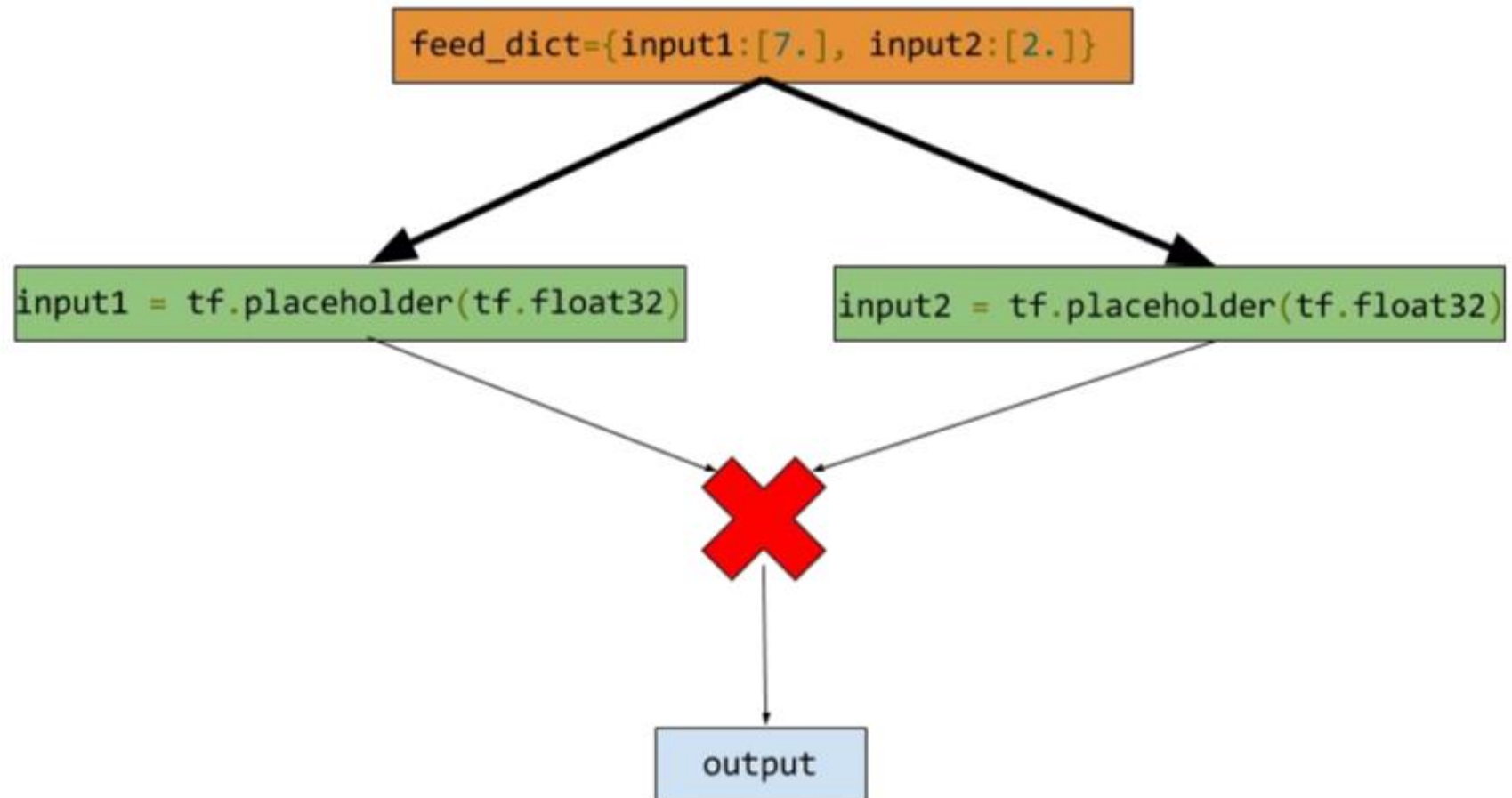
```
[array([ 14.], dtype=float32)]
```

← Define `tf.placeholder` objects for data entry.

↖ Fetch value of output from computation graph.

↖ Feed data into computation graph.

# Placeholders and Feed Dictionaries



# How to use the variable exactly.

- `tf.variable_scope()` provides simple name-spacing to avoid clashes.
- `tf.get_variable()` creates/accesses variables from within a variable scope.

```
with tf.variable_scope("foo"):
    with tf.variable_scope("bar"):
        v = tf.get_variable("v", [1])
assert v.name == "foo/bar/v:0"
```

```
with tf.variable_scope("foo"):
    v = tf.get_variable("v", [1])
    tf.get_variable_scope().reuse_variables()
    v1 = tf.get_variable("v", [1])
assert v1 == v
```

# Understanding get\_variable()

Behavior depends on whether variable reuse enabled.

- Case 1: reuse set to false — Create and return new variable

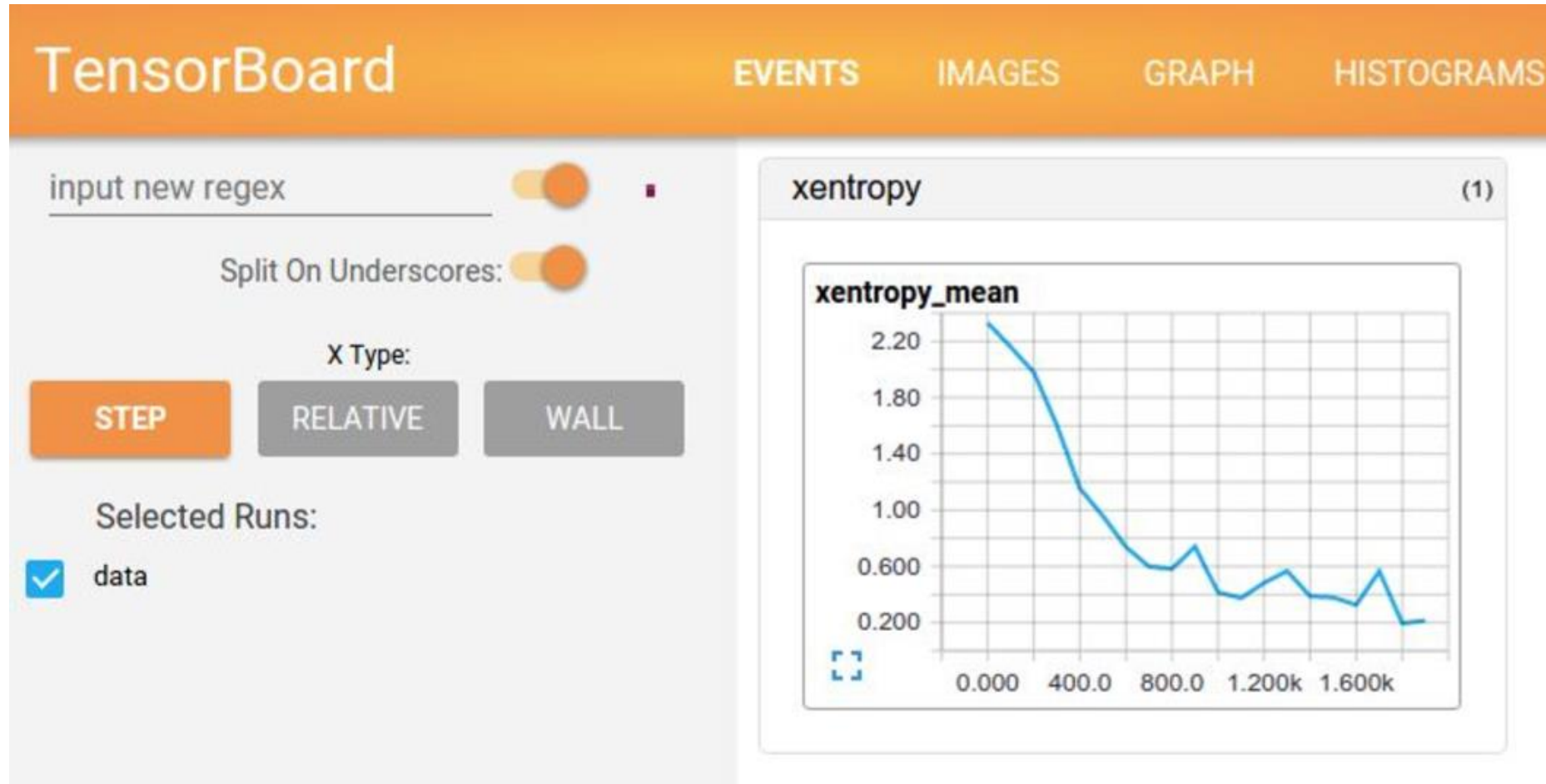
```
with tf.variable_scope("foo"):
    v = tf.get_variable("v", [1])
    assert v.name == "foo/v:0"
```

- Case 2: reuse set to true — Search for existing variable with given name. If none found Raise ValueError

```
with tf.variable_scope("foo"):
    v = tf.get_variable("v", [1])
with tf.variable_scope("foo", reuse=True):
    v1 = tf.get_variable("v", [1])
assert v1 == v
```

# TensorBoard

- [https://www.tensorflow.org/get\\_started/summaries\\_and\\_tensorboard](https://www.tensorflow.org/get_started/summaries_and_tensorboard)

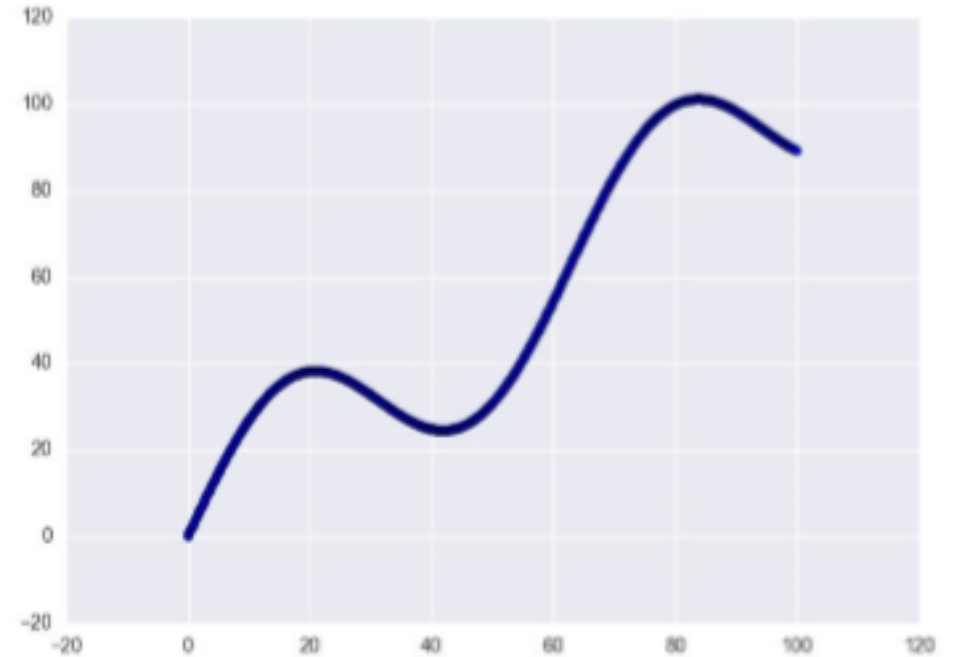


# Example of Linear Regression

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

# Define input data
X_data = np.arange(100, step=.1)
y_data = X_data + 20 * np.sin(X_data/10)

# Plot input data
plt.scatter(X_data, y_data)
plt.show()
```





# Define the placeholder


```
# Define data size and batch size
n_samples = 1000
batch_size = 100

# Tensorflow is sensitive about shapes, resize
X_data = np.reshape(X_data, (n_samples, 1))
y_data = np.reshape(y_data, (n_samples, 1))

# Define placeholders for input
X = tf.placeholder(tf.float32, shape=(batch_size, 1))
y = tf.placeholder(tf.float32, shape=(batch_size, 1))
```

# Build Linear Regression model

```
# Define variables to be learned
with tf.variable_scope("linear-regression"):
    W = tf.get_variable("weights", (1, 1),
                        initializer=tf.random_normal_initializer())
    b = tf.get_variable("bias", (1,),
                        initializer=tf.constant_initializer(0.0))
    y_pred = tf.matmul(X, W) + b
    loss = tf.reduce_sum((y - y_pred)**2 / n_samples)
```


$$J(W, b) = \frac{1}{N} \sum_{i=1}^N (y_i - (Wx_i + b))^2$$

# Define optimizer and train

```
opt = tf.train.AdamOptimizer()
opt_operation = opt.minimize(loss)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for _ in range(500):
        indices = np.random.choice(n_samples, batch_size)
        X_batch, y_batch = X_data[indices], y_data[indices]
        # Do gradient descent step
        _, loss_val = sess.run([opt_operation, loss],
                               feed_dict={X: X_batch, y: y_batch})
        print(loss_val)
```

# Resource

- <https://www.tensorflow.org/tutorials>
- <http://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf>
- [http://cs231n.stanford.edu/slides/2018/cs231n\\_2018\\_lecture08.pdf](http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture08.pdf)

# Outline

- Python
- Basic of Machine Learning
- TensorFlow
- Pytorch

# Tutorials of Pytorch (1.1.0)

- Construct calculation Graph
- Calculation operation
- Example of Linear Regression

# Dynamic Graph

- A graph is created the fly.
- Allow us to write it like numpy (can output result and continue define more calculation).

# How to build a Calculation graph

- Build Graph and Calculate

```
# Build the Calculation Graph
# and calculate
a = torch.tensor([1, 1])
b = torch.tensor([2, 2])
d = a + b
print("d:", d)
```

```
tensor([3, 3])
```

- Continue to build the graph

```
# Build the Calculation Graph
# and calculate
a = torch.tensor([1, 1])
b = torch.tensor([2, 2])
d = a + b
print("d:", d)

# Continue to build the graph
c = d * b
print("c:", c)
```

```
tensor([3, 3])
tensor([6, 6])
```



# Calculation operation

| <b>Numpy</b>                                       | <b>PyTorch</b>                                       |
|----------------------------------------------------|------------------------------------------------------|
| <code>a = np.zeros((2, 2)), np.ones((2, 2))</code> | <code>torch.zeros((2, 2)), torch.ones((2, 2))</code> |
| <code>np.sum(b, axis=1)</code>                     | <code>torch.sum(b, dim=1)</code>                     |
| <code>a.shape</code>                               | <code>b.shape</code>                                 |
| <code>np.reshape(a, (1,4))</code>                  | <code>a.view((1, 4))</code>                          |
| <code>b * 5 + 1</code>                             | <code>b * 5 + 1</code>                               |
| <code>np.dot(a,b)</code>                           | <code>a.mm(b)</code>                                 |
| <code>y = a[0, 0], a[:, 0], a[0, :]</code>         | <code>a[0, 0], a[:, 0], a[0, :]</code>               |

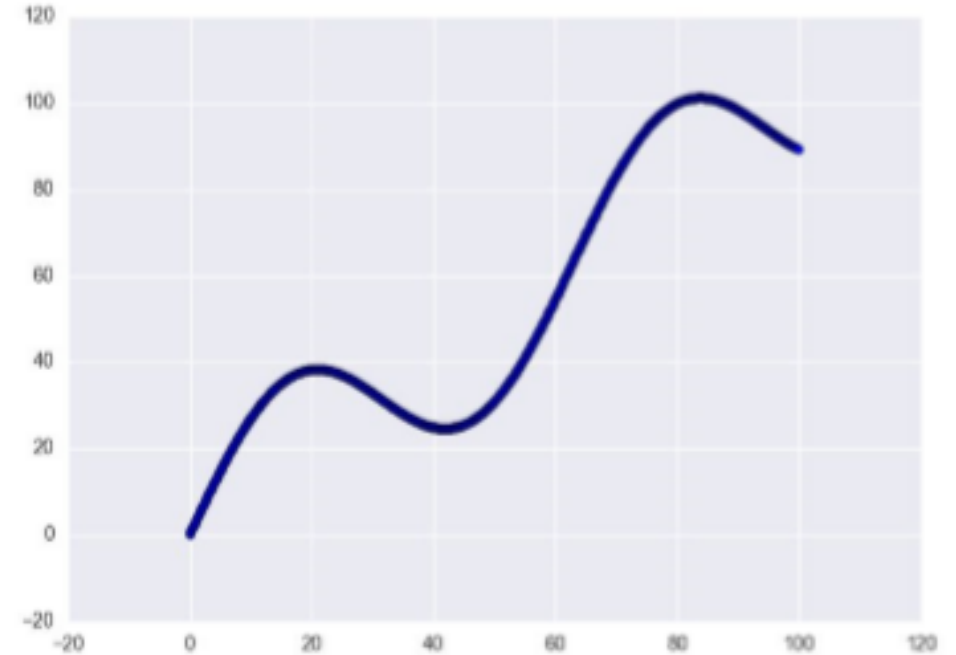
# Variable (No needed) and Tensor

- After version 0.40, tensor and variable are merged.
- So you can compute gradients on one tensor as needed, and you can update them(May be gradient descent).

# Example of Linear Regression

```
import torch
import numpy as np
import matplotlib.pyplot as plt

# Define input data
X_data = np.arange(100, step=.1, dtype='float32')
y_data = X_data + 20 * np.sin(X_data/10)
# Plot input data
plt.scatter(X_data, y_data)
plt.show()
```



# Convert input into torch tensor

```
# Define data size and batch size
n_samples = 1000
batch_size = 100

# resize
X_data = np.reshape(X_data, (n_samples, 1))
y_data = np.reshape(y_data, (n_samples, 1))

# Convert inputs and targets to PyTorch tensors
X_data = torch.from_numpy(X_data)
y_data = torch.from_numpy(y_data)
```

# Build Linear Regression model

```
# Initializing weights and biases
w = torch.randn(1, 1, requires_grad=True)
b = torch.randn(1, requires_grad=True)
# Define the model
def model(x):
    return x * w.t() + b
# MSE loss
def mse(t1, t2):
    diff = t1 - t2
    return torch.sum(diff * diff) / diff.numel()
```

```
optimizer = torch.optim.Adam(params=[w, b])
for i in range(500):
    indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_data[indices], y_data[indices]
    preds = model(X_batch)
    loss = mse(preds, y_batch)
    # Do gradient descent step
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

# Packages needed (neural network)

- torch
- torch.nn: includes module, layers...
- torch.nn.functional: includes useful function, softmax, sigmoid
- torch.optim: optimizer, like: Adam, SGD
- torchvision: Data loaders, popular model, image transformations

# Resource

- <https://pytorch.org/tutorials/>

## Deep Learning with PyTorch: A 60 Minute Blitz

Understand PyTorch's Tensor library and neural networks at a high level.

Getting Started

## Learning PyTorch with Examples

This tutorial introduces the fundamental concepts of PyTorch through self-contained examples.

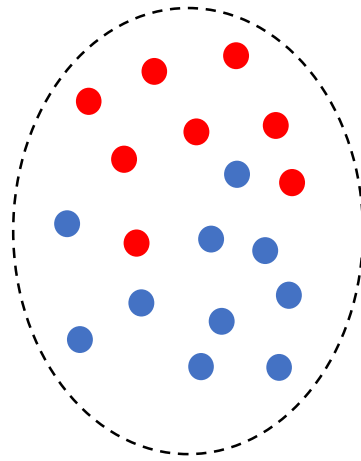
Getting Started

- <https://pytorch-cn.readthedocs.io/zh/latest/>
- <https://mofanpy.com/tutorials/>
- [http://cs231n.stanford.edu/slides/2019/cs231n\\_2019\\_lecture06.pdf](http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture06.pdf)

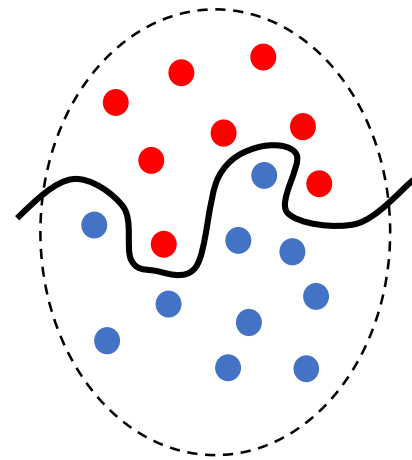


# Problems may be encountered: Overfitting

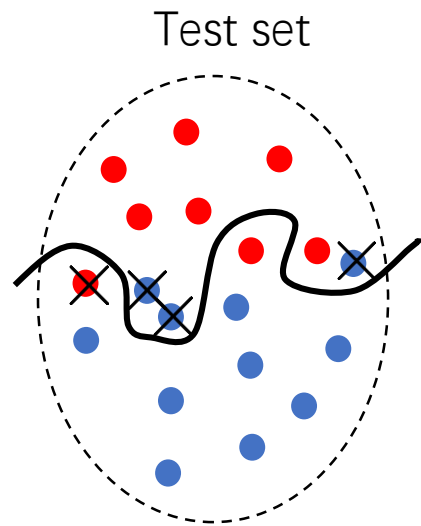
Training set



Training set

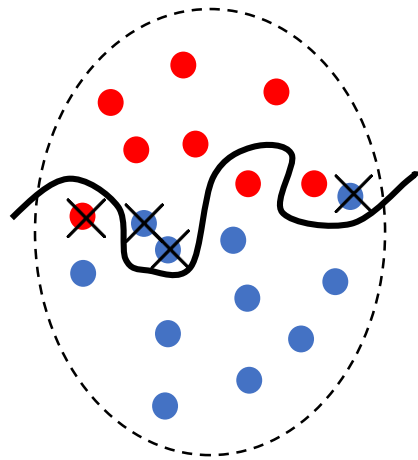


# Overfitting

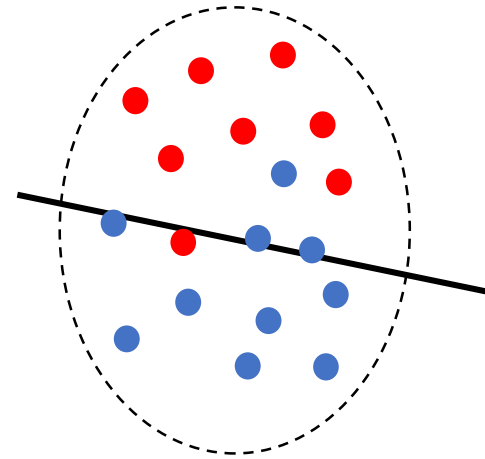


# Overfitting

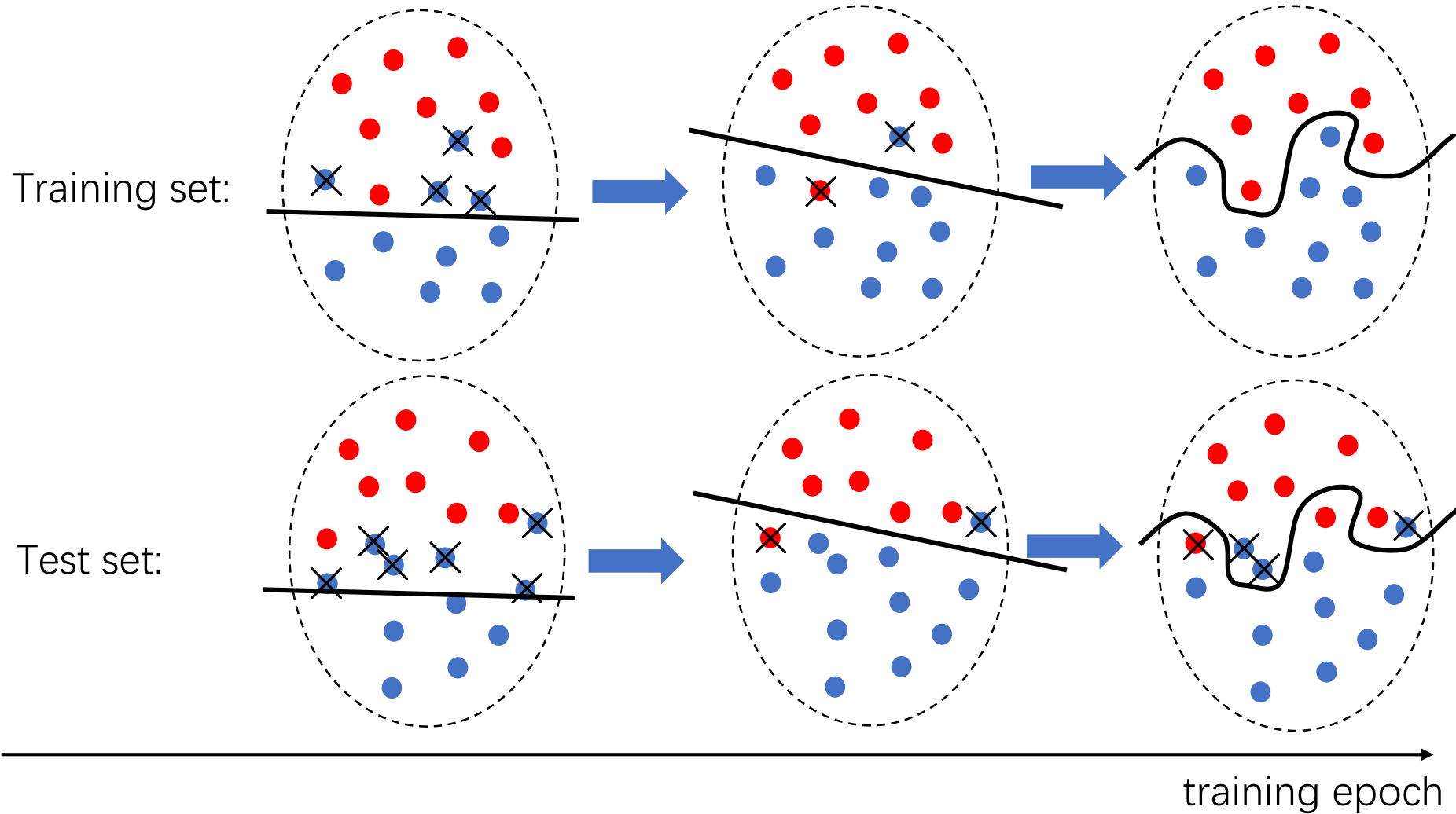
Test set



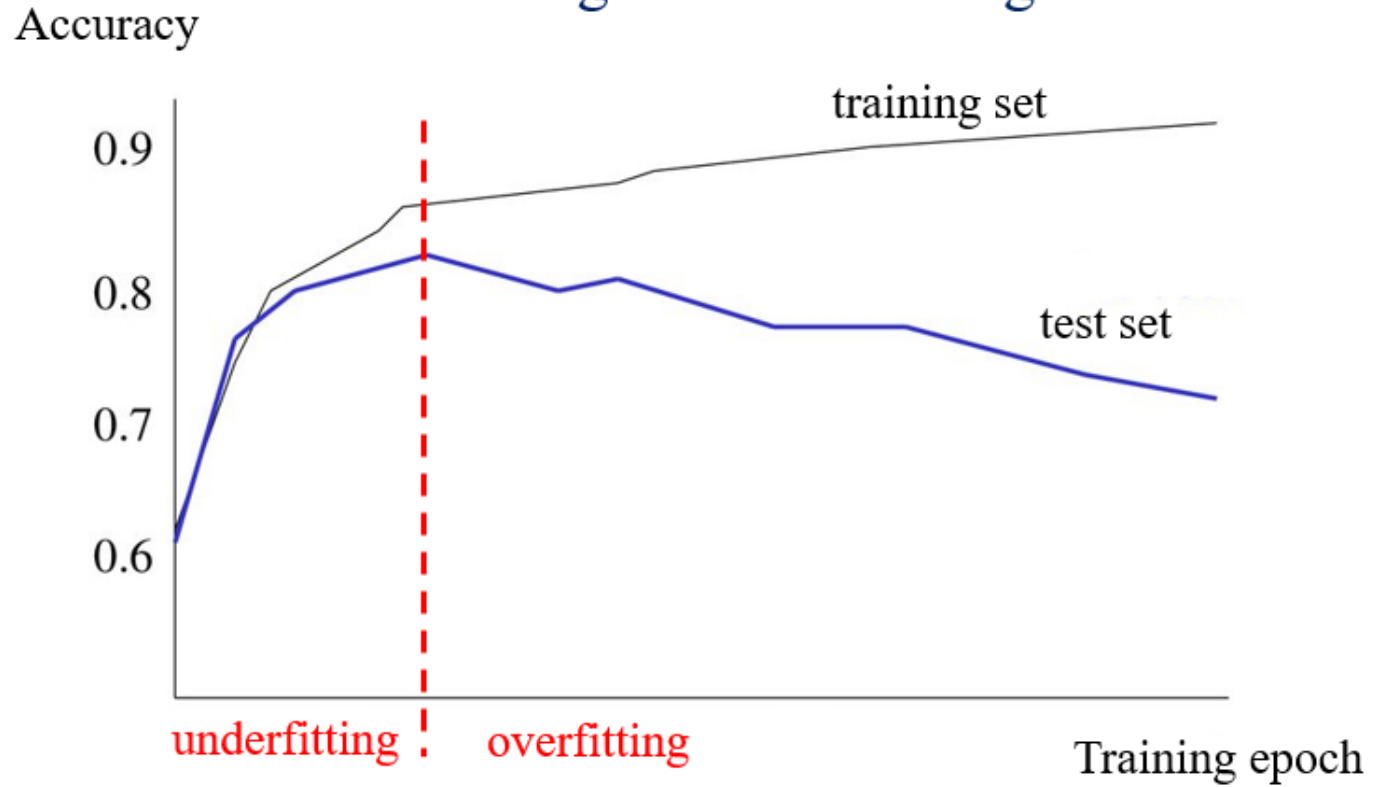
appropriate



# Overfitting and underfitting



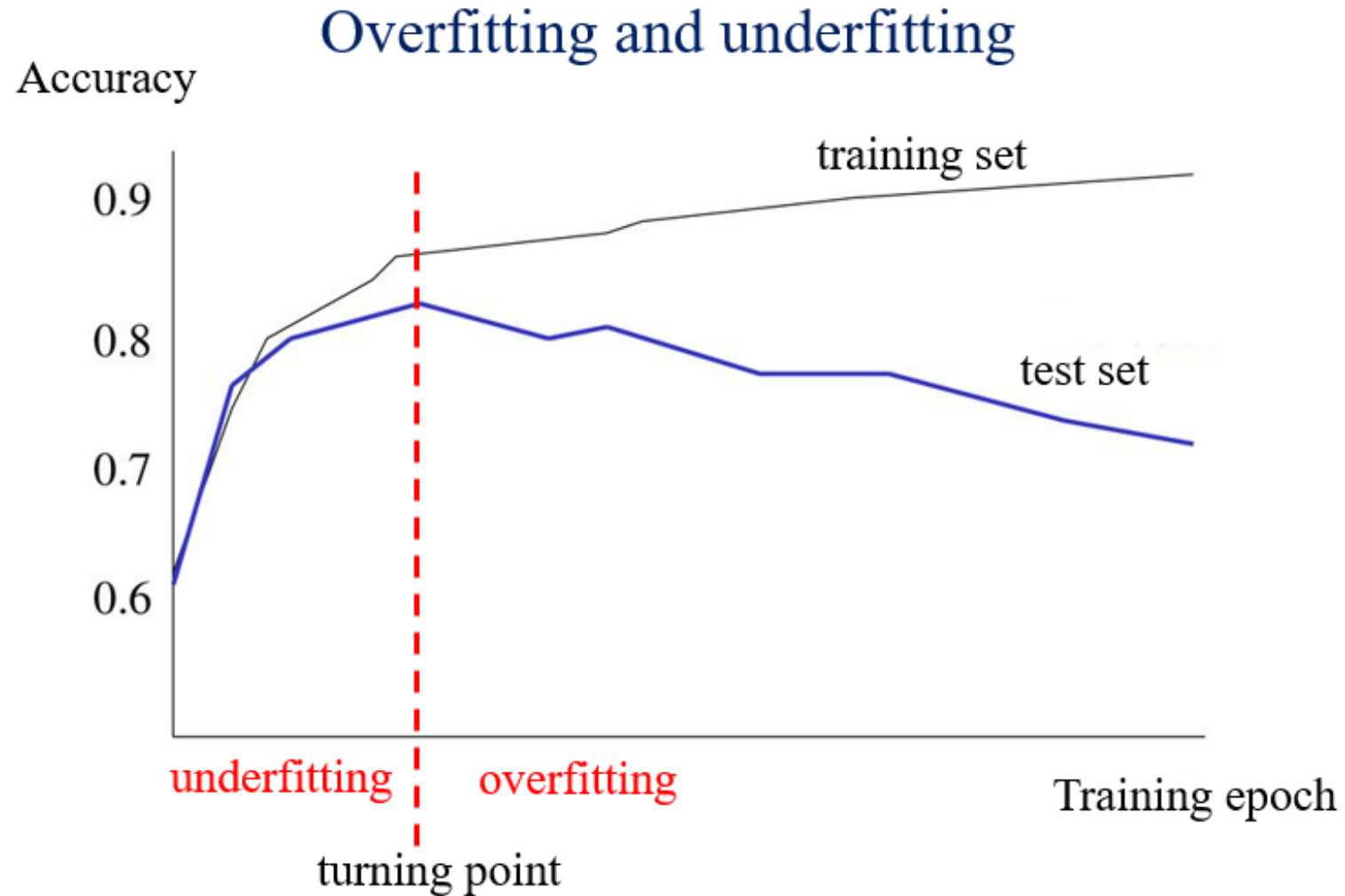
## Overfitting and underfitting



**Underfitting:** training accuracy ↑ test accuracy ↑

**Overfitting:** training accuracy ↑ test accuracy ↓

# Early stopping



# Regularization

- Limit the parameters in the model so that the parameters of the model are not too large.
- Avoid any one parameter playing a dominant role.

$$J(W, b) = \frac{1}{N} \sum_{i=1}^N (y_i - (Wx_i + b))^2 + 0.0001W^2 + 0.0001b^2$$

**THANKS!**