# Mainstream Development Frameworks in the Industry

## Foreword

- This chapter describes:
  - Definition of deep learning framework and its advantages, and two mainstream deep learning frameworks PyTorch and TensorFlow
  - Basic operations and common modules of TensorFlow 2.x (by focusing on code)
  - MNIST handwritten digit recognition experiment performed based on TensorFlow for deeply understanding and getting familiar with a deep learning modeling process

## Objectives

On completion of this course, you will be able to:

- Describe a deep learning framework.
- Know mainstream deep learning frameworks.
- Know the features of PyTorch.
- Know the features of TensorFlow.
- Differentiate between TensorFlow 1.x and 2.x.
- Master the basic syntax and common modules of TensorFlow 2.x.
- Master the process of an MNIST handwritten digit recognition experiment.

## Contents

# Deep Learning Framework

- A deep learning framework is an interface, library or a tool which allows us to build deep learning models more easily and quickly, without getting into the details of underlying algorithms. A deep learning framework can be regarded as a set of building blocks. Each component in the building blocks is a model or algorithm. Therefore, developers can use components to assemble models that meet requirements, and do not need to start from scratch.
- The emergence of deep learning frameworks lowers the requirements for developers. Developers no longer need to compile code starting from complex neural networks and back-propagation algorithms. Instead, they can use existing models to configure parameters as required, where the model parameters are automatically trained. Moreover, they can add self-defined network layers to the existing models, or select required classifiers and optimization algorithms directly by invoking existing code.

theano
Caffe
dmlc
mxnet
DL4J Deep Learning for Java
PYTORCH
torch
A SCIENTIFIC COMPUTING FRAMEWORK FOR LUAJIT
TensorFlow
MindSpore

# Contents

## PyTorch

- PyTorch is a Python-based machine learning computing framework developed by Facebook. It is developed based on Torch, a scientific computing framework supported by a large number of machine learning algorithms. Torch is a tensor operation library similar to NumPy, featured by high flexibility, but is less popular because it uses the programming language Lua. This is why PyTorch is developed.
- In addition to Facebook, institutes such as Twitter, GMU, and Salesforce also use PyTorch.

Image source: [http://PyTorch123.com/FirstSection/PyTorchIntro/](http://PyTorch123.com/FirstSection/PyTorchIntro/)

# Features of PyTorch

- **Python first**: PyTorch does not simply bind Python to a C++ framework. PyTorch directly supports Python access at a fine grain. Developers can use PyTorch as easily as using NumPy or SciPy. This not only lowers the threshold for understanding Python, but also ensures that the code is basically consistent with the native Python implementation.
- **Dynamic neural network**: Many mainstream frameworks such as TensorFlow 1.x do not support this feature. To run TensorFlow 1.x, developers must create static computational graphs in advance, and run the **feed** and **run** commands to repeatedly execute the created graphs. In contrast, PyTorch with this feature is free from such complexity, and PyTorch programs can dynamically build/adjust computational graphs during execution.
- **Easy to debug**: PyTorch can generate dynamic graphs during execution. Developers can stop an interpreter in a debugger and view output of a specific node.
- PyTorch provides tensors that support CPUs and GPUs, greatly accelerating computing.

## Contents

## TensorFlow

- TensorFlow is Google's second-generation open-source software library for digital computing. The TensorFlow computing framework supports various deep learning algorithms and multiple computing platforms, ensuring high system stability.

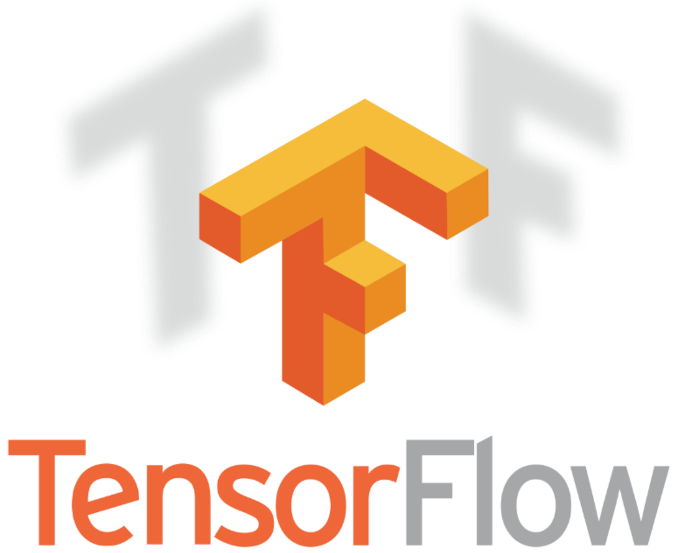


Image source: [https://www.TensorFlow.org/](https://www.TensorFlow.org/)

# Features of TensorFlow

Scalability

Multi-lingual

GPU

Multi-platform

Powerful computing

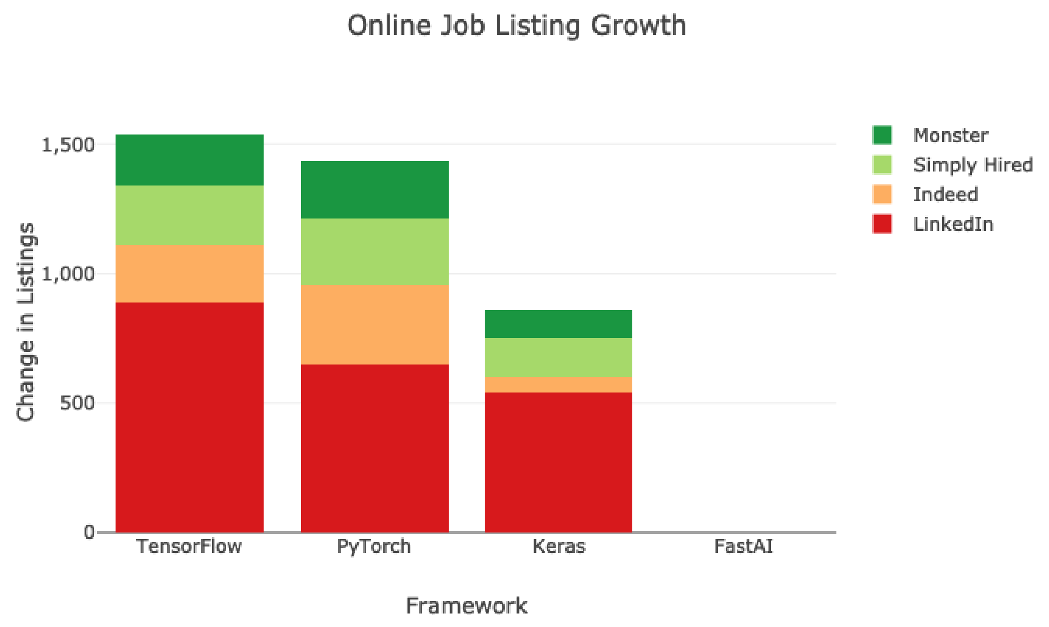
Distributed

## TensorFlow - Distributed

- TensorFlow can run on different computers:
  - From smartphones to computer clusters, to generate desired training models.
- Currently, supported native distributed deep learning frameworks include only TensorFlow, CNTK, Deeplearning4J, and MXNet.
- When a single GPU is used, most deep learning frameworks rely on cuDNN, and therefore support almost the same training speed, provided that the hardware computing capabilities or allocated memories slightly differ. However, for large-scale deep learning, massive data makes it difficult for the single GPU to complete training in a limited time. To handle such cases, TensorFlow enables distributed training.

# Why TensorFlow?

- TensorFlow is considered as one of the best libraries for neural networks, and can reduce difficulty in deep learning development. In addition, as it is open-source, it can be conveniently maintained and updated, thus the efficiency of development can be improved.
- Keras, ranking third in the number of stars on GitHub, is packaged into an advanced API of TensorFlow 2.0, which makes TensorFlow 2.x mor flexible, and easier to debug.



Demand on the recruitment market

# TensorFlow 2.x vs. TensorFlow 1.x

- Disadvantages of TensorFlow 1.0:
  - After a tensor is created in TensorFlow 1.0, the result cannot be returned directly. To obtain the result, the session mechanism needs to be created, which includes the concept of graph, and code cannot run without session.run. This style is more like the hardware programming language VHDL.
  - Compared with some simple frameworks such as PyTorch, TensorFlow 1.0 adds the session and graph concepts, which are inconvenient for users.
  - It is complex to debug TensorFlow 1.0, and its APIs are disordered, making it difficult for beginners. Learners will come across many difficulties in using TensorFlow 1.0 even after gaining the basic knowledge. As a result, many researchers have turned to PyTorch.

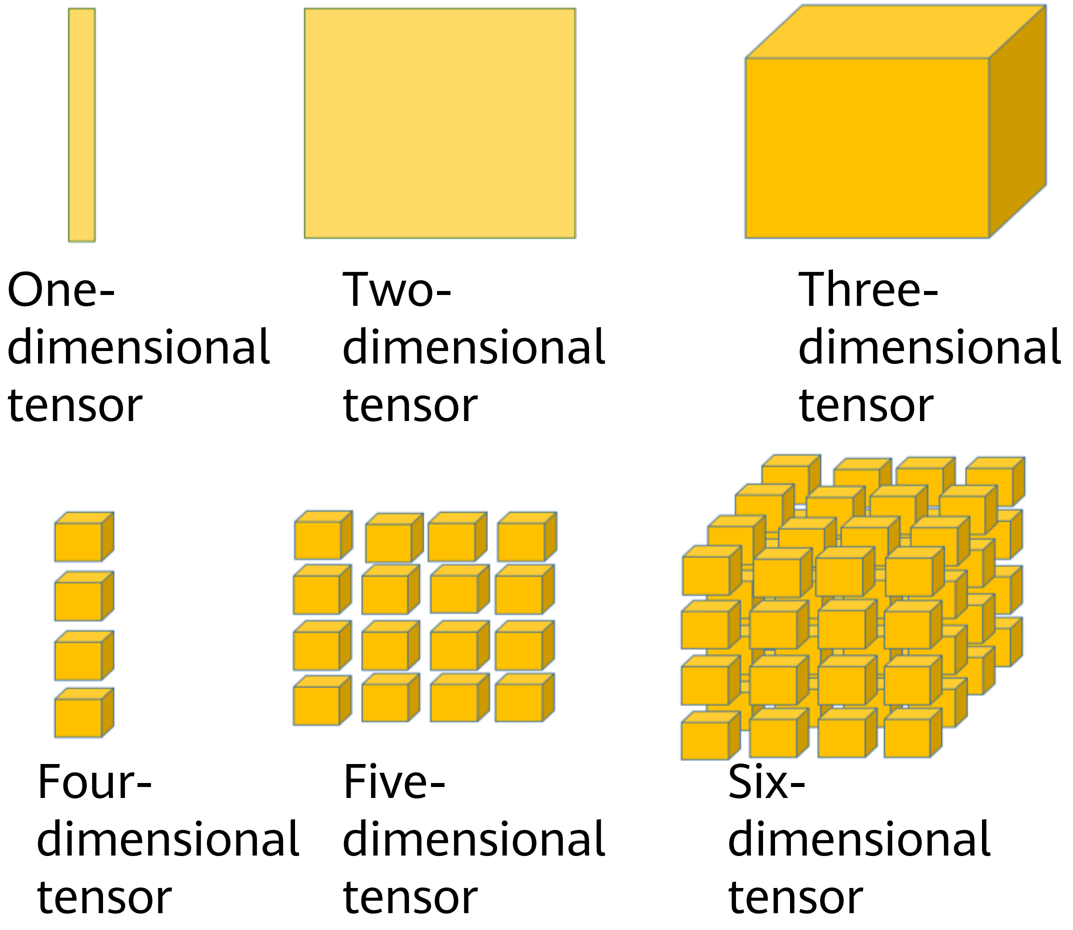
## TensorFlow 2.x vs. TensorFlow 1.x

- Features of TensorFlow 2.x:
  - Advanced API Keras:
    - Easy to use: The graph and session mechanisms are removed. What you see is what you get, just like Python and PyTorch.
  - Major improvements:
    - The core function of TensorFlow 2.x is the dynamic graph mechanism called eager execution. It allows users to compile and debug models like normal programs, making TensorFlow easier to learn and use.
    - Multiple platforms and languages are supported, and compatibility between components can be improved via standardization on exchange formats and alignment of APIs.
    - Deprecated APIs are deleted and duplicate APIs are reduced to avoid confusion.
    - Compatibility and continuity: TensorFlow 2.x provides a module enabling compatibility with TensorFlow 1.x.
    - The tf.contrib module is removed. Maintained modules are moved to separate repositories. Unused and unmaintained modules are removed.

# Contents

# Tensors

- Tensors are the most basic data structures in TensorFlow. All data is encapsulated in tensors.
- Tensor: a multidimensional array
  - A scalar is a rank-0 tensor. A vector is a rank-1 tensor. A matrix is a rank-2 tensor.
- In TensorFlow, tensors are classified into:
  - Constant tensors
  - Variable tensors



## Basic Operations of TensorFlow 2.x

- The following describes common APIs in TensorFlow by focusing on code. The main content is as follows:
  - Methods for creating constants and variables
  - Tensor slicing and indexing
  - Dimension changes of tensors
  - Arithmetic operations on tensors
  - Tensor concatenation and splitting
  - Tensor sorting

# Eager Execution Mode of TensorFlow 2.x

- Static graph: TensorFlow 1.x using static graphs (graph mode) separates computation definition and execution by using computational graphs. This is a declarative programming model. In graph mode, developers need to build a computational graph, start a session, and then input data to obtain an execution result.
- Static graphs are advantageous in distributed training, performance optimization, and deployment, but inconvenient for debugging. Executing a static graph is similar to invoking a compiled C language program, and internal debugging cannot be performed in this case. Therefore, eager execution based on dynamic computational graphs emerges.
- Eager execution is a command-based programming method, which is the same as native Python. A result is returned immediately after an operation is performed.

## AutoGraph

- Eager execution is enabled in TensorFlow 2.x by default. Eager execution is intuitive and flexible for users (easier and faster to run a one-time operation), but may compromise performance and deployability.
- To achieve optimal performance and make a model deployable anywhere, you can run @tf.function to add a decorator to build a graph from a program, making Python code more efficient.
- tf.function can build a TensorFlow operation in the function into a graph. In this way, this function can be executed in graph mode. Such practice can be considered as encapsulating the function as a TensorFlow operation of a graph.

# Contents

## Common Modules of TensorFlow 2.x (1)

- tf: Functions in the tf module are used to perform common arithmetic operations, such as tf.abs (calculating an absolute value), tf.add (adding elements one by one), and tf.concat (concatenating tensors). Most operations in this module can be performed by NumPy.
- tf.errors: error type module of TensorFlow
- tf.data: implements operations on datasets.
  - Input pipes created by tf.data are used to read training data. In addition, data can be easily input from memories (such as NumPy).
- tf.distributions: implements various statistical distributions.
  - The functions in this module are used to implement various statistical distributions, such as Bernoulli distribution, uniform distribution, and Gaussian distribution.

## Common Modules of TensorFlow 2.x (2)

- tf.io.gfile: implements operations on files.
  - Functions in this module can be used to perform file I/O operations, copy files, and rename files.
- tf.image: implements operations on images.
  - Functions in this module include image processing functions. This module is similar to OpenCV, and provides functions related to image luminance, saturation, phase inversion, cropping, resizing, image format conversion (RGB to HSV, YUV, YIQ, or gray), rotation, and sobel edge detection. This module is equivalent to a small image processing package of OpenCV.
- tf.keras: a Python API for invoking Keras tools.
  - This is a large module that enables various network operations.

# Keras Interface

- TensorFlow 2.x recommends Keras for network building. Common neural networks are included in **Keras.layers**.
- Keras is a high-level API used to build and train deep learning models. It can be used for rapid prototype design, advanced research, and production. It has the following three advantages:
  - Easy to use

    Keras provides simple and consistent GUIs optimized for common cases. It provides practical and clear feedback on user errors.
  - Modular and composable

    You can build Keras models by connecting configurable building blocks together, with little restriction.
  - Easy to extend

    You can customize building blocks to express new research ideas, create layers and loss functions, and develop advanced models.

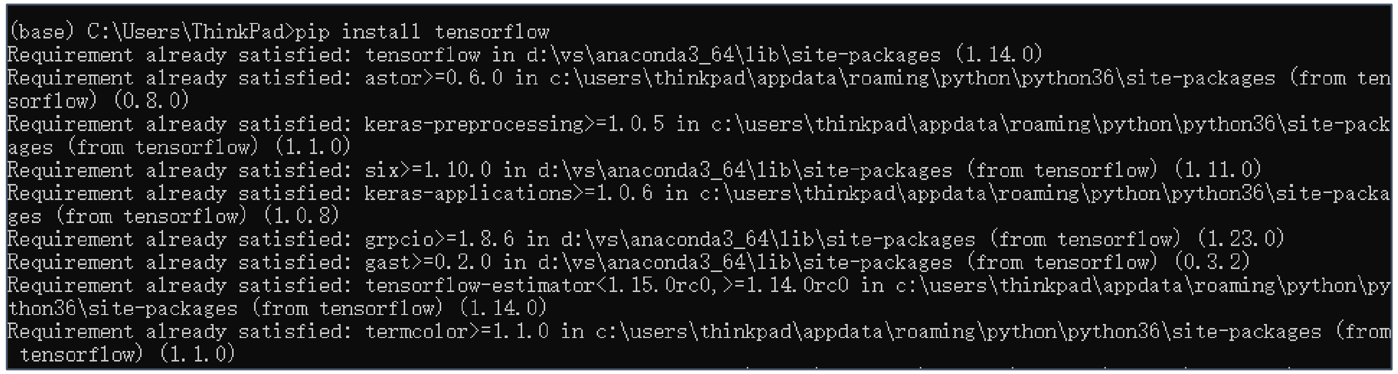
## Common Keras Methods and Interfaces

- The following describes common methods and interfaces of tf.keras by focusing on code. The main content is as follows:
  - Dataset processing: datasets and preprocessing
  - Neural network model creation: Sequential, Model, Layers…
  - Network compilation: compile, Losses, Metrics, and Optimizers
  - Network training and evaluation: fit, fit_generator, and evaluate

# Contents

## TensorFlow Environment Setup in Windows 10

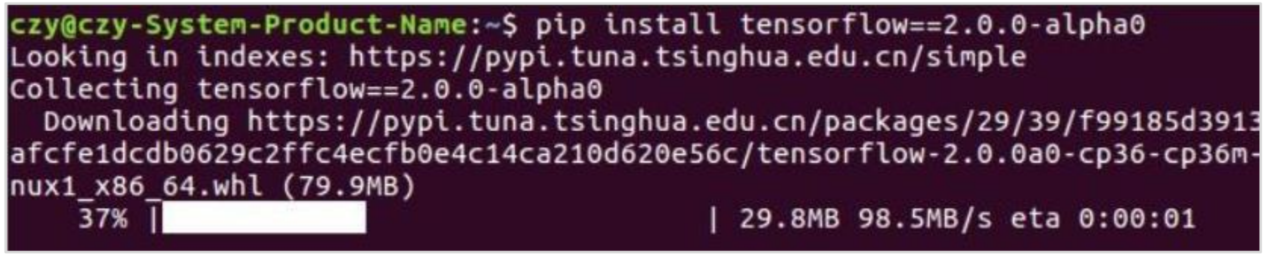
- Environment setup in Windows 10:
  - Operating system: Windows 10
  - pip software built in Anaconda 3 (adapting to Python 3)
  - TensorFlow installation:
    - Open Anaconda Prompt and run the pip command to install TensorFlow.
    - Run pip install TensorFlow in the command line interface.



```
(base) C:\Users\ThinkPad>pip install tensorflow
Requirement already satisfied: tensorflow in d:\vs\anaconda3_64\lib\site-packages (1.14.0)
Requirement already satisfied: astor>=0.6.0 in c:\users\thinkpad\appdata\roaming\python\python36\site-packages (from ten
sorflow) (0.8.0)
Requirement already satisfied: keras-preprocessing>=1.0.5 in c:\users\thinkpad\appdata\roaming\python\python36\site-pack
ages (from tensorflow) (1.1.0)
Requirement already satisfied: six>=1.10.0 in d:\vs\anaconda3_64\lib\site-packages (from tensorflow) (1.11.0)
Requirement already satisfied: keras-applications>=1.0.6 in c:\users\thinkpad\appdata\roaming\python\python36\site-packa
ges (from tensorflow) (1.0.8)
Requirement already satisfied: grpcio>=1.8.6 in d:\vs\anaconda3_64\lib\site-packages (from tensorflow) (1.23.0)
Requirement already satisfied: gast>=0.2.0 in d:\vs\anaconda3_64\lib\site-packages (from tensorflow) (0.3.2)
Requirement already satisfied: tensorflow-estimator<1.15.0rc0,>=1.14.0rc0 in c:\users\thinkpad\appdata\roaming\python\py
thon36\site-packages (from tensorflow) (1.14.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\thinkpad\appdata\roaming\python\python36\site-packages (from
 tensorflow) (1.1.0)
```

## TensorFlow Environment Setup in Ubuntu/Linux

- The simplest way for installing TensorFlow in Linux is to run the pip command.

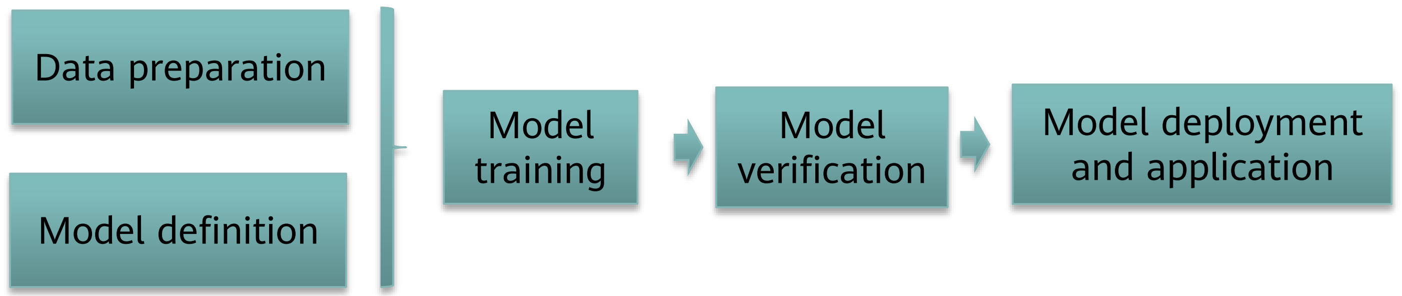


```
czy@czy-System-Product-Name:~$ pip install tensorflow==2.0.0-alpha0
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting tensorflow==2.0.0-alpha0
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/29/39/f99185d3913
afcfe1dcdb0629c2ffc4ecfb0e4c14ca210d620e56c/tensorflow-2.0.0a0-cp36-cp36m-
nux1_x86_64.whl (79.9MB)
    37% |████████████                              | 29.8MB 98.5MB/s eta 0:00:01
```

- pip command: pip install TensorFlow==2.1.0

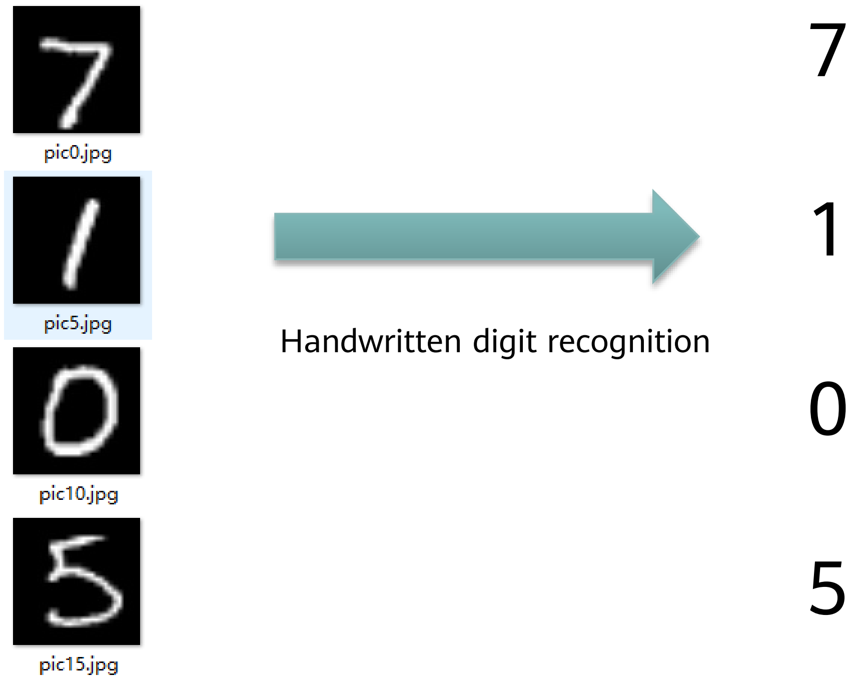
# TensorFlow Development Process

- Data preparation
  - Data exploration
  - Data processing
- Network construction
  - Defining a network structure.
  - Defining loss functions, selecting optimizers, and defining model evaluation indicators.
- Model training and verification
- Model saving
- Model restoration and invoking



## Project Description

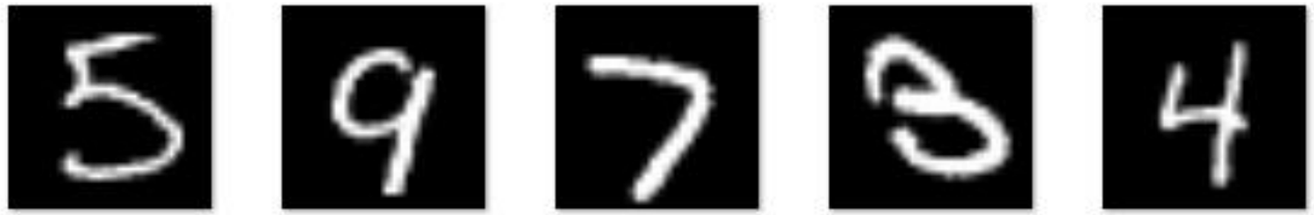
- Handwritten digit recognition is a common image recognition task where computers recognize text in handwriting images. Different from printed fonts, handwriting of different people has different sizes and styles, making it difficult for computers to recognize handwriting. This project applies deep learning and TensorFlow tools to train and build models based on the MNIST handwriting dataset.



# Data Preparation

- MNIST datasets
  - Download the MNIST datasets from http://yann.lecun.com/exdb/mnist/.
  - The MNIST datasets consist of a training set and a test set.
    - Training set: 60,000 handwriting images and corresponding labels
    - Test set: 10,000 handwriting images and corresponding labels

**Examples**



**Corresponding labels**

[0,0,0,0,0, 1,0,0,0,0]    [0,0,0,0,0, 0,0,0,0,1]    [0,0,0,0,0, 0,0,1,0,0]    [0,0,0,1,0, 0,0,0,0,0]    [0,0,0,0,1, 0,0,0,0,0]

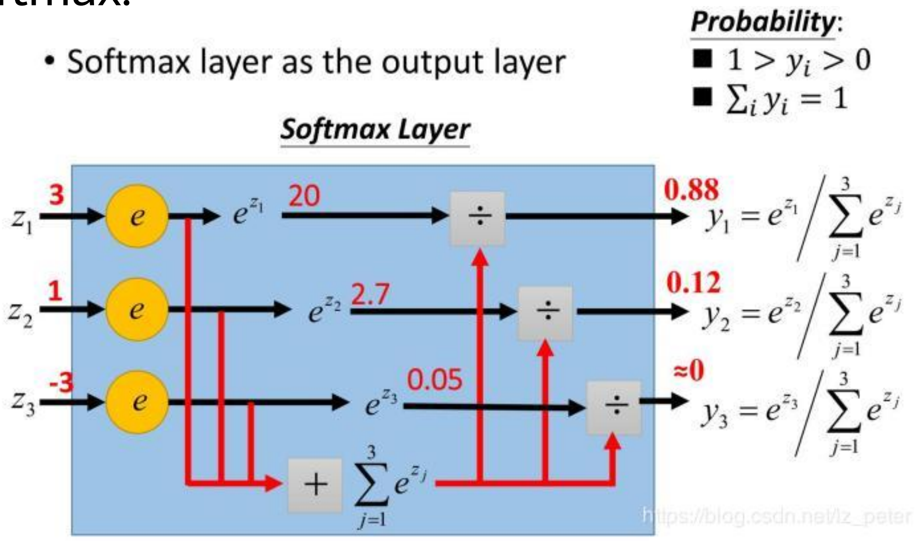
# Network Structure Definition (1)

- Softmax regression model

$$evidence_i = \sum_j W_{i,j} x_j + b_i$$

$$y = softmax(evidence)$$

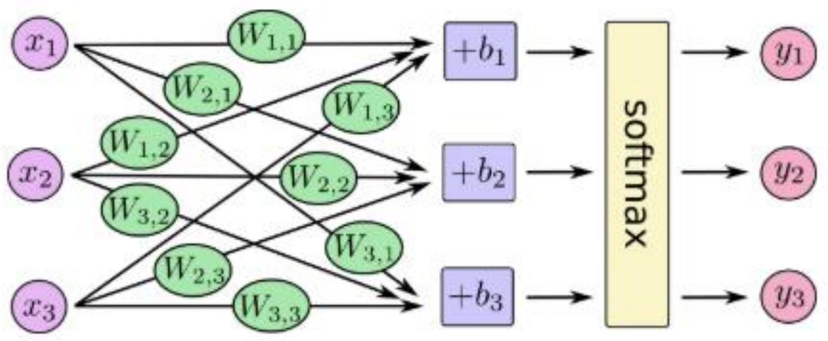
- The softmax function is also called normalized exponential function. It is a derivative of the binary classification function sigmoid in terms of multi-class classification. The following figure shows the calculation method of softmax.



• Softmax layer as the output layer
Probability:
■ $1 > y_i > 0$
■ $\sum_i y_i = 1$
Softmax Layer
$z_1$
3
$z_2$
1
$z_3$
-3
e
$e^{z_1}$
20
$e^{z_2}$
2.7
$e^{z_3}$
0.05
+
$\sum_{j=1}^{3} e^{z_j}$
÷
0.88
$y_1 = e^{z_1} / \sum_{j=1}^{3} e^{z_j}$
0.12
$y_2 = e^{z_2} / \sum_{j=1}^{3} e^{z_j}$
≈0
$y_3 = e^{z_3} / \sum_{j=1}^{3} e^{z_j}$
https://blog.csdn.net/lz_peter

# Network Structure Definition (2)

- The process of model establishment is the core process of network structure definition.
- The network operation process defines how model output is calculated based on input.



- Matrix multiplication and vector addition are used to express the calculation process of softmax.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left( \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

## Network Structure Definition (3)

- TensorFlow-based softmax regression model

```
## import tensorflow
import tensorflow as tf
##define input variables with operator symbol variables.
‘’’ we use a variable to feed data into the graph through the placeholders X. Each input
image is flattened into a 784-dimensional vector. In this case, the shape of the tensor is
[None, 784], None indicates can be of any length. ’’’
X = tf.placeholder(tf.float32,[None,784])
‘’’ The variable that can be modified is used to indicate the weight w and bias b. The initial
values are set to 0. ’’’
w = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
‘’’ If tf.matmul(x, w) is used to indicate that x is multiplied by w, the Soft regression
equation is y = softmax(wx+b)'‘’
y = tf.nn.softmax(tf.matmul(x,w)+b)
```

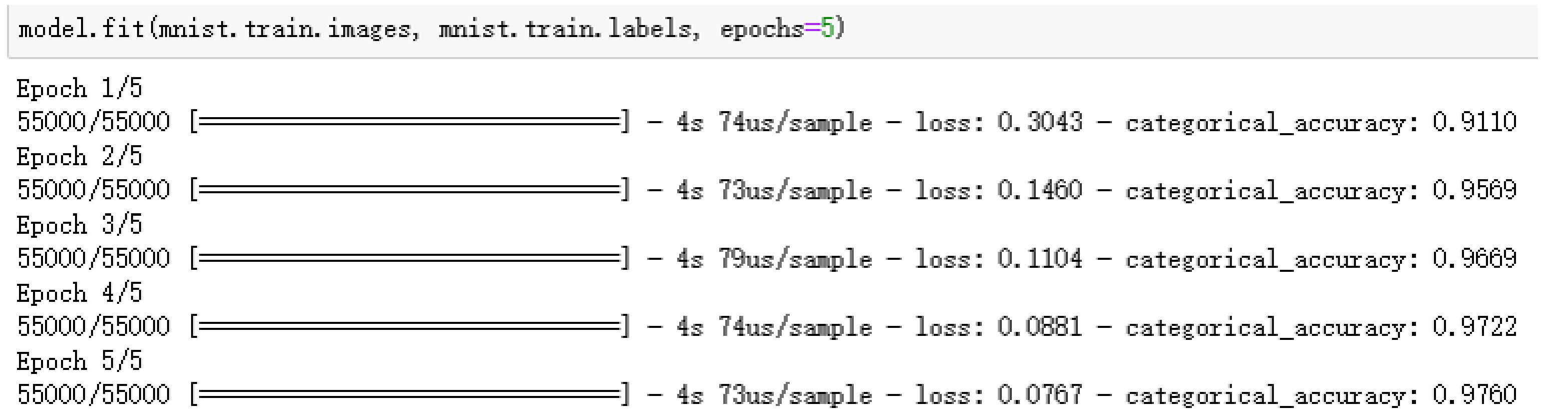
# Network Compilation

- Model compilation involves the following two parts:
  - Loss function selection
- In machine learning/deep learning, an indicator needs to be defined to indicate whether a model is proper. This indicator is called cost or loss, and is minimized as far as possible. In this project, the cross entropy loss function is used.
  - Gradient descent method
- A loss function is constructed for an original model needs to be optimized by using an optimization algorithm, to find optimal parameters and further minimize a value of the loss function. Among optimization algorithms for solving machine learning parameters, the gradient descent-based optimization algorithm (Gradient Descent) is usually used.

```
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=[tf.keras.metrics.categorical_accuracy])
```

# Model Training

- Training process:
  - All training data is trained through batch iteration or full iteration. In the experiment, all data is trained five times.
  - In TensorFlow, model.fit is used for training, where epoch indicates the number of training iterations.

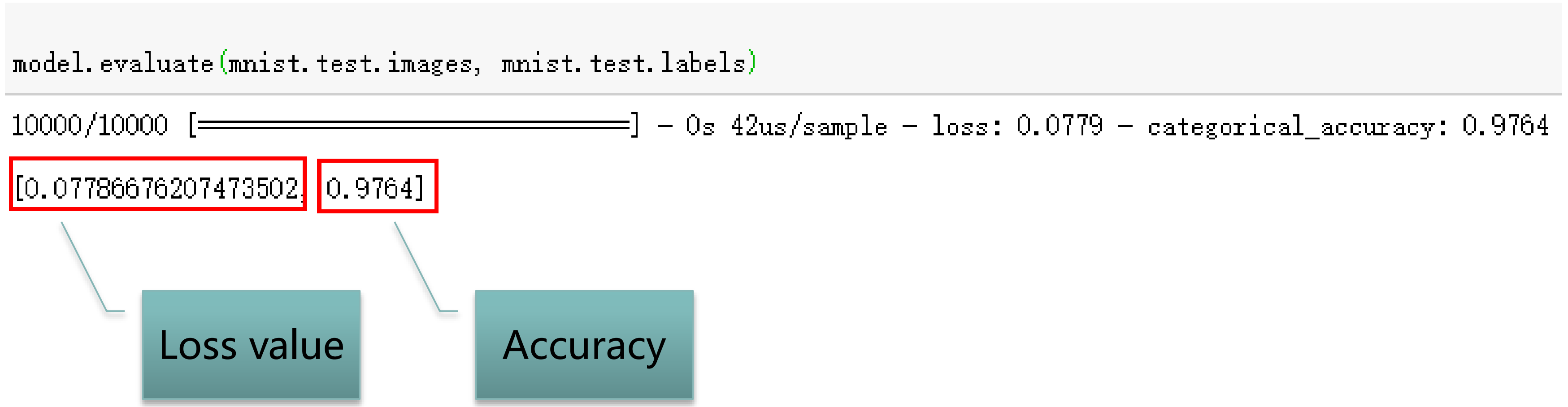


```
model.fit(mnist.train.images, mnist.train.labels, epochs=5)
```

```
Epoch 1/5
55000/55000 [==============================] - 4s 74us/sample - loss: 0.3043 - categorical_accuracy: 0.9110
Epoch 2/5
55000/55000 [==============================] - 4s 73us/sample - loss: 0.1460 - categorical_accuracy: 0.9569
Epoch 3/5
55000/55000 [==============================] - 4s 79us/sample - loss: 0.1104 - categorical_accuracy: 0.9669
Epoch 4/5
55000/55000 [==============================] - 4s 74us/sample - loss: 0.0881 - categorical_accuracy: 0.9722
Epoch 5/5
55000/55000 [==============================] - 4s 73us/sample - loss: 0.0767 - categorical_accuracy: 0.9760
```

# Model Evaluation

- You can test the model using the test set, compare predicted results with actual ones, and find correctly predicted labels, to calculate the accuracy of the test set.



```
model.evaluate(mnist.test.images, mnist.test.labels)
```

```
10000/10000 [==============================] - 0s 42us/sample - loss: 0.0779 - categorical_accuracy: 0.9764
[0.07786676207473502, 0.9764]
```

## Summary

- This chapter describes the following content by focusing on code: Features of common deep learning frameworks, including PyTorch and TensorFlow Basic syntax and common modules of TensorFlow 2.x Development procedure of TensorFlow.