2. TensorFlow

2.1. What is TensorFlow

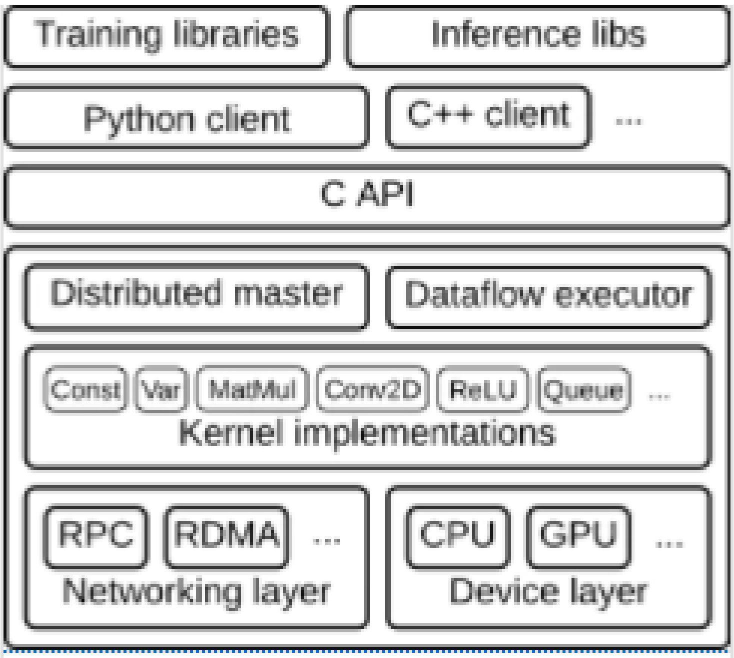
2.1.1. Definition

TensorFlow™ is a symbolic mathematical system based on dataflow programming (dataflow programming), which is widely used in the programming implementation of various machine learning (machine learning) algorithms. Its predecessor is Google's neural network algorithm library DistBelief.

Tensorflow has a multi-layer structure and can be deployed on various servers, PC terminals and web pages. It also supports GPU and TPU high-performance numerical computing. It is widely used in Google's internal product development and scientific research in various fields.

2.1.2 Core Components

The core components (core runtime) of distributed TensorFlow include: distributed master, dataflow executor/worker service, kernel implementation, and the bottom device layer (device layer)/network layer (networking layer).



TensorFlow的代码结构 [13]

1) Distributed master

The distribution center cuts subgraphs from the input data flow graph, divides them into operation fragments and starts the executor. When processing the data flow graph, the distribution center performs preset operation optimizations, including common subexpression elimination, constant folding, etc.

2) The executor is responsible for running graph operations in processes and devices, and sending and receiving results from other executors. Distributed TensorFlow has a parameter server to aggregate and update model parameters returned by other executors. When scheduling local devices, the executor will choose to perform parallel computing and GPU acceleration.

3) The kernel application is responsible for a single graph operation, including mathematical calculations, array operations, control flow, and state management operations. The kernel application uses Eigen to perform parallel computing of tensors, cuDNN library to perform GPU acceleration, and gemmlowp to perform low numerical precision calculations. In addition, users can register additional kernels (fused kernels) in the kernel application to improve the efficiency of basic operations, such as activation functions and their gradient calculations.

2.2, TensorFlow 2

2.2.1, Introduction

TensorFlow is an open source deep learning tool released by Google in November 2015. We can use it to **quickly build** deep neural networks and **train deep learning models**. The main purpose of using TensorFlow and other open source frameworks is to provide us with a **module toolbox that is more conducive to building deep learning networks, so that the code can be simplified during development, and the final model presented is more concise and easy to understand.

2.2.2, Upgrade direction

1), Use Keras and Eager Execution to easily build models.

2), Achieve robust production environment model deployment on any platform.

3), Provide powerful experimental tools for research.

4), Simplify the API by cleaning up abandoned APIs and reducing duplication.

2.3, TensorFlow basic concept syntax

2.3.1, Tensor

1), Tensor is the core data unit of TensorFlow, which is essentially an array of any dimension. We call a 1-dimensional array a vector, a 2-dimensional array a matrix, and a tensor can be regarded as an N-dimensional array.

2), In TensorFlow, each Tensor has two basic attributes: data type (default: float32) and shape. The data types are roughly shown in the following table,

Tensor type	Description
tf.float32	32-bit floating point number
tf.float64	64-bit floating point number
tf.int64	64-bit signed integer
tf.int32	32-bit signed integer
tf.int16	16-bit signed integer
tf.int8	8-bit signed integer
tf.uint8	8-bit unsigned integer
tf.string	Variable-length byte array
tf.bool	Boolean
tf.complex64	Real and imaginary numbers

3) According to different uses, there are mainly 2 types of tensors in TensorFlow, namely

- tf.Variable: variable Tensor, which requires the initial value to be specified, is often used to define variable parameters, such as the weights of a neural network.
- tf.constant: constant Tensor, you need to specify the initial value, define the unchanged tensor

4), define a variable Tensor

Create a new python file, name it Tensor_Variable, and then give it execution permissions,

```
sudo chmod a+x Tensor_Variable.py
```

Paste the following code into it,

```
import tensorflow as tf
v = tf.Variable([[1, 2], [3, 4]]) # Two-dimensional variable with shape (2, 2)
print(v)
```

Run the test,

```
python3 Tensor_Variable.py
```

Note: ROSMASTER must use python3 to use tensorflow2.0 or above

Output,

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=int32, numpy=
array([[1, 2],
[3, 4]], dtype=int32)>
```

5) Define a constant Tensor

Create a new python file, name it Tensor_constant, and then give it execution permissions,

```
sudo chmod a+x Tensor_constant.py
```

Paste the following code into it,

```
import tensorflow as tf
v = tf.constant([[1, 2], [3, 4]]) # Two-dimensional variable with shape (2, 2)
print(v)
```

Run the test,

```
python3 Tensor_constant.py
```

Output,

```
<tf.Tensor: id=9, shape=(2, 2), dtype=int32, numpy=
array([[1, 2],
[3, 4]], dtype=int32)>
```

If you look closely, you will find that the output tensor has three attributes: shape, data type dtype, and NumPy array.

6) Commonly used methods for creating special constant tensors:

- tf.zeros: Create a constant Tensor with a specified shape and all 0s

Example: c = tf.zeros([2, 2]) # 2x2 constant Tensor with all 0s

Output,

```
<tf.Tensor: id=12, shape=(2, 2), dtype=float32, numpy=
array([[0., 0.],
[0., 0.]], dtype=float32)
```

- tf.ones_like: Create a constant Tensor with all 1s based on a certain shape

Example: v = tf.ones_like(c) # Create a constant Tensor with all 1s in the same shape as tensor c. **Note that the shape here refers to the attribute shape of the tensor**

Output,

```
<tf.Tensor: id=15, shape=(3, 3), dtype=float32, numpy=
array([[1., 1.],
[1., 1.]],dtype=float32)
```

- tf.fill: Create a constant Tensor with a specified shape and all scalar values

Example: a = tf.fill([2, 3], 6) # 2x3 constant Tensor with all 6s

Output,

```
<tf.Tensor: id=18, shape=(2, 3), dtype=int32, numpy=
array([[6, 6, 6],
[6, 6, 6]], dtype=int32)>
```

- tf.linspace: Create an equally spaced sequence

Example: c = tf.linspace(1.0, 10.0, 5, name="linspace")

Output,

```
<tf.Tensor: id=22, shape=(5,), dtype=float32, numpy=array([ 1.  , 3.25, 5.5 ,
7.75, 10. ], dtype=float32)>
```

- tf.range: Create a sequence of numbers

Example: c = tf.range(start=2, limit=8, delta=2)

Output,

```
<tf.Tensor: id=26, shape=(5,), dtype=int32, numpy=array([2, 4, 6, 8],
dtype=int32)>
```

For this part of the code, please refer to:

```
~/TensorFlow_demo/tensor_init.py
```

2.3.2. Use the Eager Execution (dynamic graph) mechanism to operate on tensors

1) Changes in the tensor operation mechanism between TensorFlow2 and TensorFlow1.x

The dynamic graph mechanism is the biggest difference between TensorFlow2.x and TensorFlow1.x. This is similar to PyThorch, simplifying the code and execution process.

2) Take tensor addition as an example,

Create a new python file, name it tensor_plus, and then give it execution permissions,

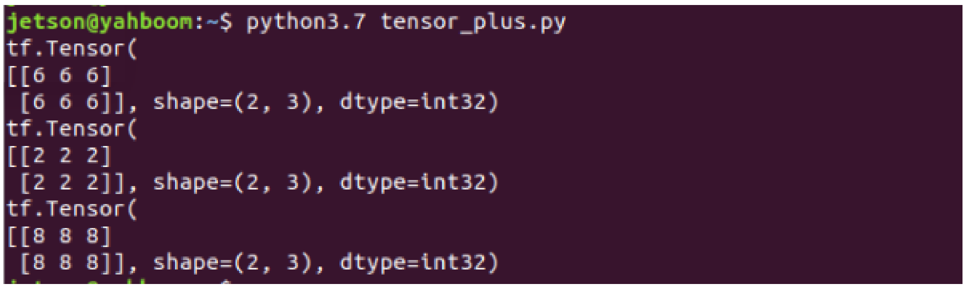
```
sudo chmod a+x tensor_plus.py
```

Paste the following code into it,

```
a = tf.fill([2, 3], 6)
b = tf.fill([2, 3], 2)
c = a + b
print(a)
print(b)
print(c)
```

Run the test,

```
python3 tensor_plus.py
```



```
jetson@yahboom:~$ python3.7 tensor_plus.py
tf.Tensor(
[[6 6 6]
 [6 6 6]], shape=(2, 3), dtype=int32)
tf.Tensor(
[[2 2 2]
 [2 2 2]], shape=(2, 3), dtype=int32)
tf.Tensor(
[[8 8 8]
 [8 8 8]], shape=(2, 3), dtype=int32)
```

It can be found that the execution process of this python is the same, and if it is in Tersorflow1.x, you also need to establish a session, and the session executes the addition operation inside.

3), Common APIs of TensorFlow:

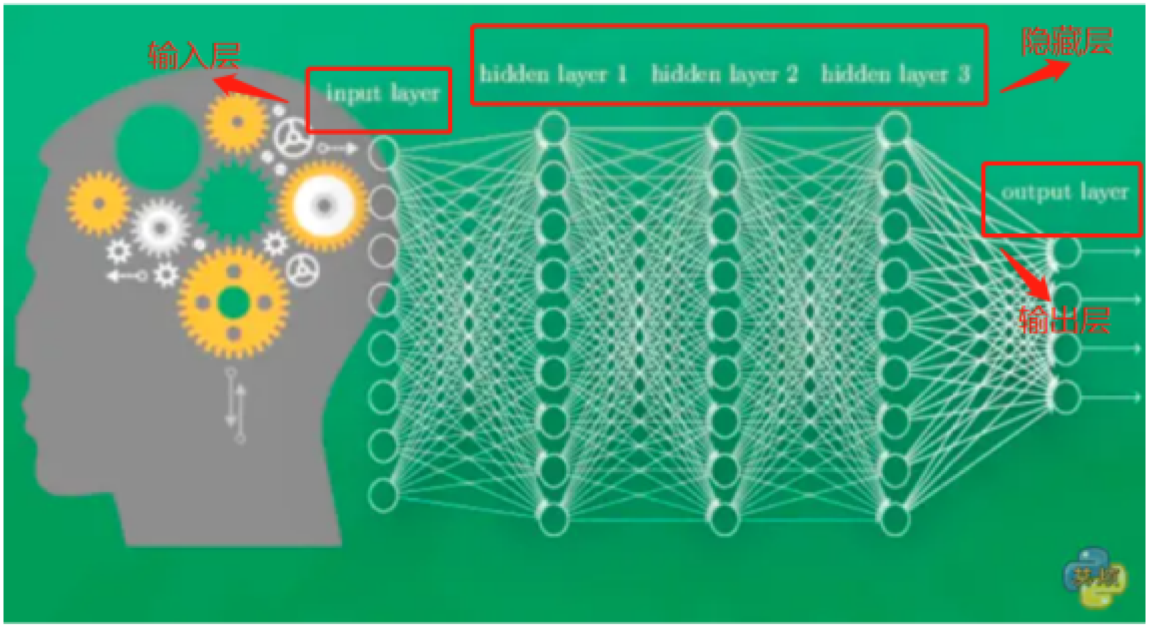
- tf.math: Mathematical calculation module, providing a large number of mathematical calculation functions.
- tf.linalg: Linear algebra module, providing a large number of linear algebra calculation methods and classes.
- tf.image: Image processing module, providing classes such as image cropping, transformation, encoding, decoding, etc.
- tf.train: Provides components for training, such as optimizers, learning rate decay strategies, etc.
- tf.nn: Provides underlying functions for building neural networks to help implement various functional layers of deep neural networks.
- tf.keras: The original Keras framework high-level API. Contains the original tf.layers medium and high-order neural network layers.
- tf.data: Input data processing module, providing classes such as tf.data.Dataset for encapsulating input data, specifying batch size, etc.

For usage of these common APIs, please refer to the official documentation:

Module: tf | TensorFlow Core v2.8.0 (google.cn)

2.3.3 Neural Network

1) A neural network is a mathematical model that exists in the computer's nervous system. It is composed of a large number of neurons that are connected and perform calculations. Based on external information, it changes the internal structure and is often used to model the complex relationship between input and output. A basic neural network structure has an input layer, a hidden layer, and an output layer. The following figure is a neural network diagram.



2) Input layer: receiving sensory information;

3) Hidden layer: processing input information;

4) Output layer: outputting the computer's understanding of input information.

2.3.4. Building and training a neural network (**kear**)

1) Import data

```
x_train = datasets.load_iris().data
y_train = datasets.load_iris().target
```

For more information about data, please refer to: tf.data.Dataset | TensorFlow Core v2.8.0 (google.cn)

2) Define a structural network that describes a neural network

```
model = tf.keras.models.Sequential()
```

Example:

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, activation='softmax',
kernel_regularizer=tf.keras.regularizers.l2())
])
```

The input parameters represent the network structure from the input layer to the output layer, which generally have the following three:

- Flatten layer: tf.keras.layers.Flatten()

For reference, please refer to the official document: tf.keras.layers.Flatten | TensorFlow Core v2.8.0 (google.cn)

- Fully connected layer: tf.keras.layers.Dense()

For reference, please refer to the official document: tf.keras.layers.Dense | TensorFlow Core v2.8.0 (google.cn)

- Convolutional layer: tf.keras.layers.Conv2D()

For reference, please refer to the official document: tf.keras.layers.Conv2D | TensorFlow Core v2.8.0 (google.cn)

3) Configure the training method for training neural networks

```
model.compile( optimizer = 优化器, loss = 损失函数, metrics = [“准确率”])
```

Example:

```
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),

 loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=[tf.keras.metrics.sparse_categorical_accuracy])
```

The input parameters are composed of the following three parts:

- optimizer: optimizer

Mainly set the learning rate lr, learning decay rate decay and momentum parameters.

- loss: loss function
- metrics: accuracy

Multiple accuracy rates can be specified.

For the specific values of the three parameters, please refer to: tf.keras.Model | TensorFlow Core v2.8.0 (google.cn)

4) Execute the training process

```
model.fit (x = input features of the training set, y = labels of the training
set, batch_size = specifies the number of samples contained in each batch when
performing gradient descent, epochs = the value when training ends,
validation_data = (input features of the test set, labels of the test set),
validation_split = how much proportion of the training set is divided into the
test set, validation_freq = the number of epoch intervals for the test)
```

Example:

```
model.fit(x_train, y_train, batch_size=32, epochs=5, validation_data=(x_test,
y_test), validation_freq=1)
```

For specific parameter settings, please refer to: [tf.keras.Model | TensorFlow Core v2.8.0 (google.cn)]()

5), print network structure and parameter statistics

```
model.summary()
```

For specific parameter settings, please refer to:[tf.keras.Model | TensorFlow Core v2.8.0 (google.cn)]()

2.3.4. Training Neural Network Example - Classic Training Cat and Dog Image Example

1) Code Path Reference

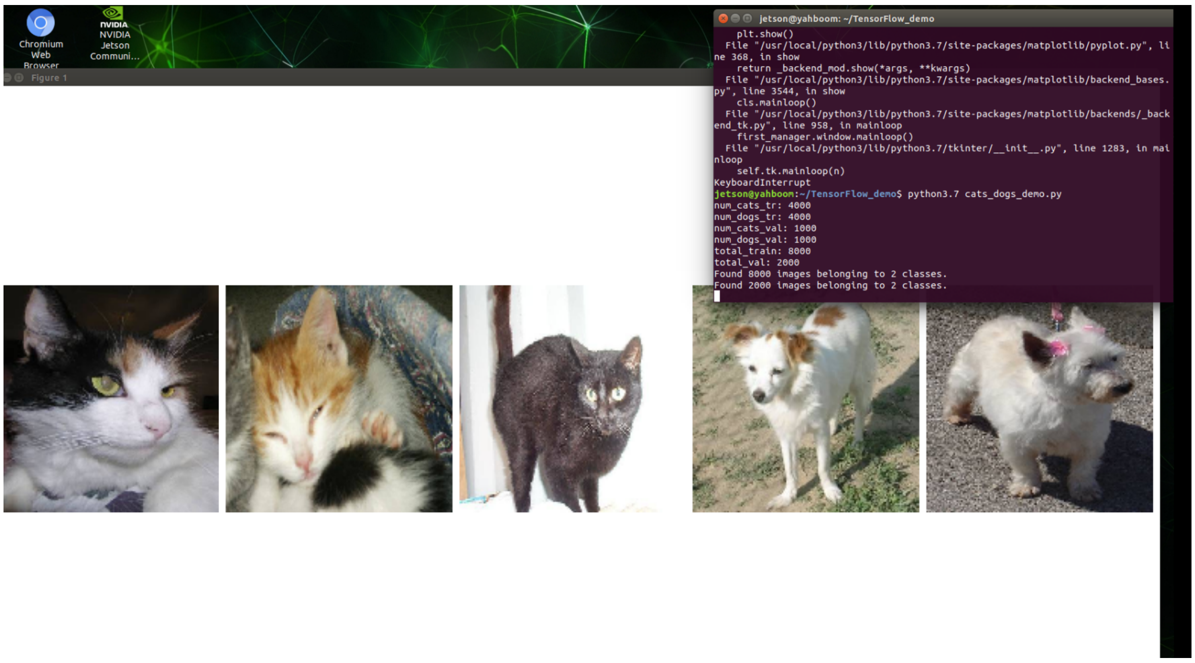
```
~/TensorFlow_demo/cats_dogs_demo.py
```

2) Run the program

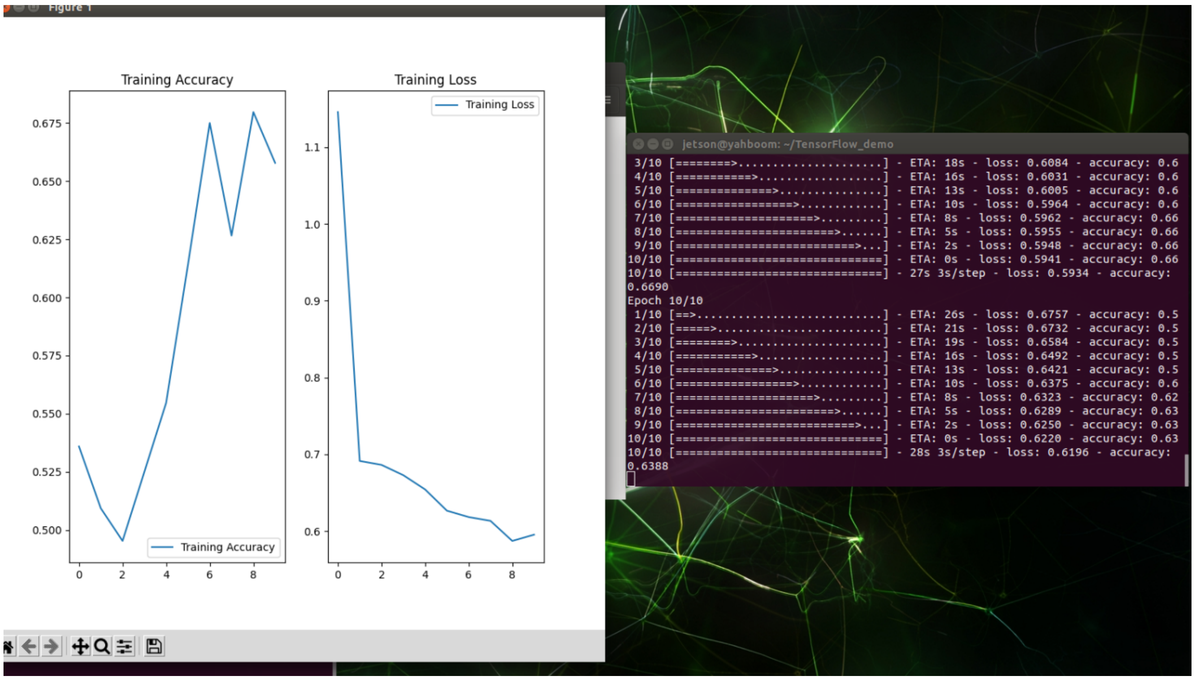
```
cd TensorFlow_demo/
python3 cats_dogs_demo.py
```

3) Screenshot of program running

When the kitten and puppy photos appear, press the q key in the picture display window to continue executing the program.



```
    plt.show()
  File "/usr/local/python3/lib/python3.7/site-packages/matplotlib/pyplot.py", li
ne 368, in show
    return _backend_mod.show(*args, **kwargs)
  File "/usr/local/python3/lib/python3.7/site-packages/matplotlib/backend_bases.
py", line 3544, in show
    cls.mainloop()
  File "/usr/local/python3/lib/python3.7/site-packages/matplotlib/backends/_back
end_tk.py", line 958, in mainloop
    first_manager.window.mainloop()
  File "/usr/local/python3/lib/python3.7/tkinter/__init__.py", line 1283, in mai
nloop
    self.tk.mainloop(n)
KeyboardInterrupt
jetson@yahboom:~/TensorFlow_demo$ python3.7 cats_dogs_demo.py
num_cats_tr: 4000
num_dogs_tr: 4000
num_cats_val: 1000
num_dogs_val: 1000
total_train: 8000
total_val: 2000
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```



```
 3/10 [========>.....................] - ETA: 18s - loss: 0.6084 - accuracy: 0.6
 4/10 [===========>..................] - ETA: 16s - loss: 0.6031 - accuracy: 0.6
 5/10 [==============>...............] - ETA: 13s - loss: 0.6005 - accuracy: 0.6
 6/10 [=================>............] - ETA: 10s - loss: 0.5964 - accuracy: 0.6
 7/10 [====================>.........] - ETA: 8s - loss: 0.5962 - accuracy: 0.66
 8/10 [=======================>......] - ETA: 5s - loss: 0.5955 - accuracy: 0.66
 9/10 [==========================>...] - ETA: 2s - loss: 0.5948 - accuracy: 0.66
10/10 [==============================] - ETA: 0s - loss: 0.5941 - accuracy: 0.66
10/10 [==============================] - 27s 3s/step - loss: 0.5934 - accuracy:
0.6690
Epoch 10/10
 1/10 [==>...........................] - ETA: 26s - loss: 0.6757 - accuracy: 0.5
 2/10 [=====>........................] - ETA: 21s - loss: 0.6732 - accuracy: 0.5
 3/10 [========>.....................] - ETA: 19s - loss: 0.6584 - accuracy: 0.5
 4/10 [===========>..................] - ETA: 16s - loss: 0.6492 - accuracy: 0.5
 5/10 [==============>...............] - ETA: 13s - loss: 0.6421 - accuracy: 0.5
 6/10 [=================>............] - ETA: 10s - loss: 0.6375 - accuracy: 0.6
 7/10 [====================>.........] - ETA: 8s - loss: 0.6323 - accuracy: 0.62
 8/10 [=======================>......] - ETA: 5s - loss: 0.6289 - accuracy: 0.63
 9/10 [==========================>...] - ETA: 2s - loss: 0.6250 - accuracy: 0.63
10/10 [==============================] - ETA: 0s - loss: 0.6220 - accuracy: 0.63
10/10 [==============================] - 28s 3s/step - loss: 0.6196 - accuracy:
0.6388
```

The model has 10 training epochs and 10 sample batches. The coordinate curve on the left shows that as the number of training times increases, the accuracy acc and error loss will rise and fall.