2. TensorFlow

2.1. What is TensorFlow?

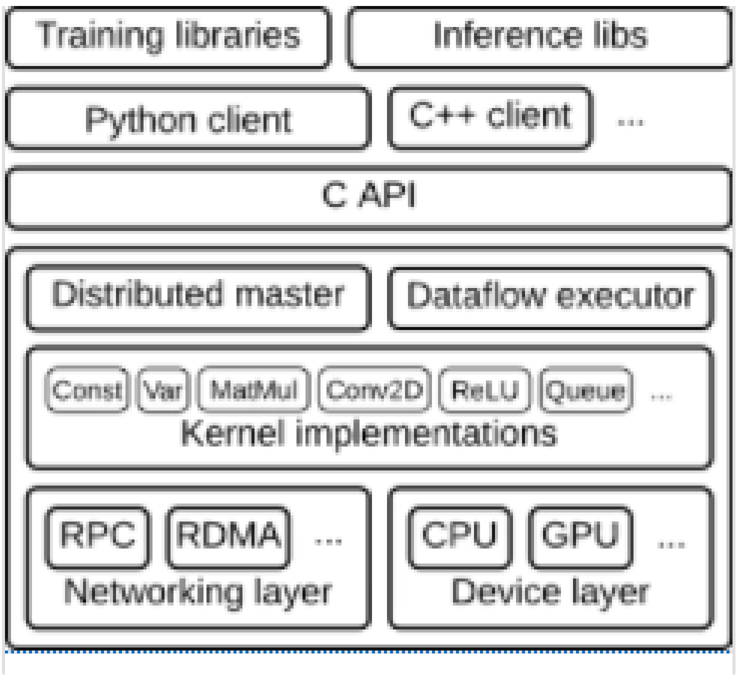
2.1.1. Definition

TensorFlow™ is a symbolic mathematics system based on data flow programming (dataflow programming) and is widely used in various types of machine learning Programming implementation of (machine learning) algorithm, its predecessor is the neural network of Google Network algorithm library DistBelief.

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

2.1.2. Core components

The core components of distributed TensorFlow (core runtime) include: distribution center (distributed master), executor (dataflow executor/worker service), kernel application (kernel implementation) and the bottom [device layer] (https://baike. baidu.com/item/Device Layer/8983070) (device layer)/Network Layer (networking layer).



1), distribution center (distributed master)

The distribution center clips subgraphs from the input data flow graph, divides them into operational fragments and starts executors. When the distribution center processes the data flow graph, it will perform preset operation optimizations, including common subexpression elimination (common subexpression elimination), constant folding (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. The executor will choose to perform parallel computing and GPU acceleration when scheduling local devices.

3. The kernel application is responsible for a single graph operation, including mathematical calculations, array manipulation, control flow and state management operations. The kernel application uses Eigen to perform parallel calculations of tensors, cuDNN libraries, etc. to perform GPU acceleration, and gemmlowp to perform low numerical precision calculations. In addition, users can perform low numerical precision calculations in the kernel application. Register additional kernels (fused kernels) 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 a deep learning open source 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 will be 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 obsolete 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 arbitrary dimensions. 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 as 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 type
tf.int32	32-bit signed integer type
tf.int16	16-bit signed integer type
tf.int8	8-bit signed integer type
tf.uint8	8-bit unsigned integer type
tf.string	Variable length byte array

Tensor type	Description
tf.bool	Boolean type
tf.complex64	Real and imaginary numbers

3). According to different uses, there are two main tensor types in TensorFlow, namely

- tf.Variable: variable Tensor, which needs to specify an initial value and is often used to define variable parameters, such as the weights of neural networks.
- tf.constant: constant Tensor, the initial value needs to be specified to define a tensor that does not change

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 inside,

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

run the test,

```
python3Tensor_Variable.py
```

Note: ROSMASTER must use python3 to properly 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 inside,

```
import tensorflow astf
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 to create new special constant tensors:

- tf.zeros: Create a new constant Tensor with the 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: Refer to a certain shape and create a new constant Tensor with all 1's

 Example: v = tf.ones_like(c) # A constant Tensor that is consistent with the shape (shape) of tensor c and is all 1. **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 new constant Tensor with a specified shape and all scalar values.

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

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 Eager Execution (dynamic graph) mechanism to perform operations on tensors

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

The dynamic graph mechanism is the biggest difference between TensorFlow2.x and TensorFlow1.x. It is similar to PyTorch and simplifies the code and execution process.

2. Take tensor addition as an example to illustrate,

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

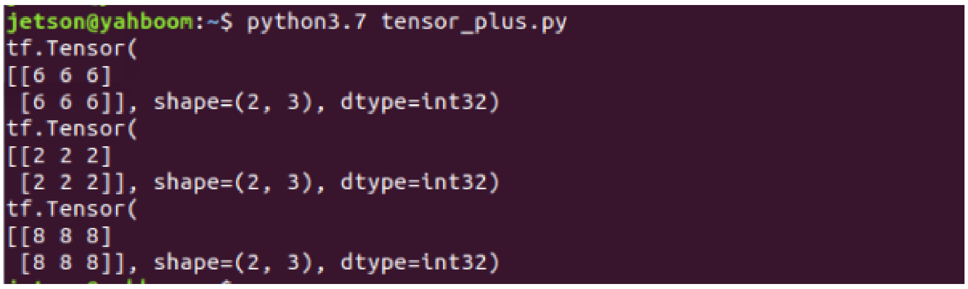
```
sudo chmod a+x tensor_plus.py
```

Paste the following code inside,

```
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, a session needs to be established, and the addition operation in the session is performed.

3. Commonly used APIs of TensorFlow:

- tf.math: Mathematical calculation module, which provides a large number of mathematical calculation functions.
- tf.linalg: Linear algebra module, which provides a large number of linear algebra calculation methods and classes.
- tf.image: Image processing module, which provides classes such as image cropping, transformation, encoding, and decoding.

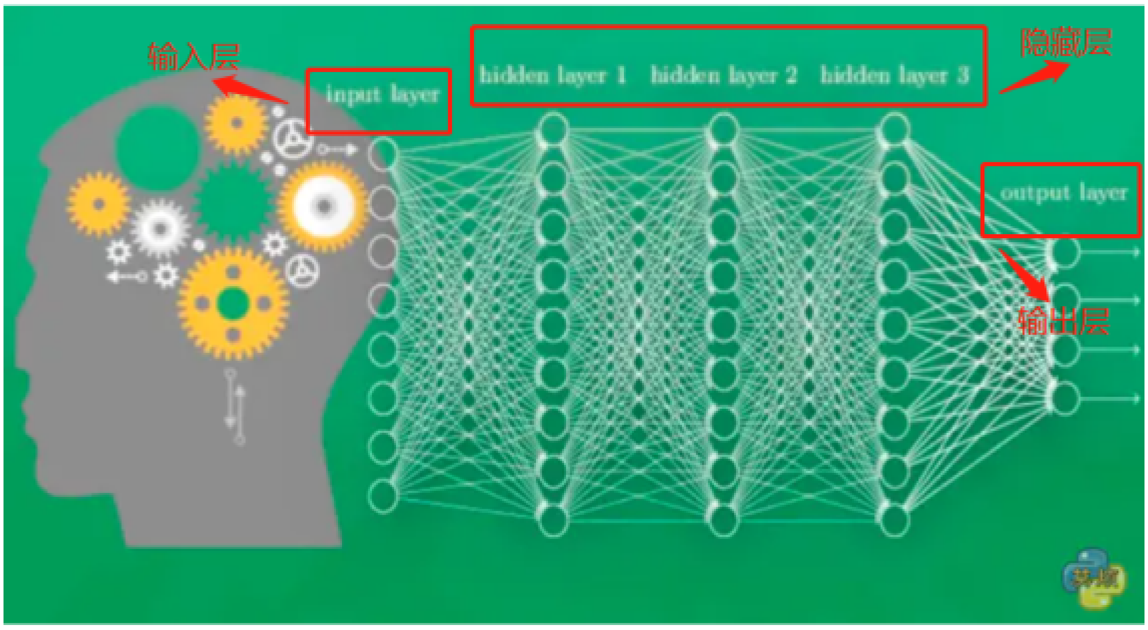
- tf.train: Provides components for training, such as optimizers, learning rate decay strategies, etc.
- tf.nn: Provides underlying functions for building neural networks to help implement various functional layers of deep neural networks.
- tf.keras: the high-level API of the original Keras framework. Contains high-order neural network layers in the original tf.layers.
- tf.data: Input data processing module, which provides classes such as tf.data.Dataset for encapsulating input data, specifying batch size, etc.

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

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

2.3.3. Neural network

1. **Neural network** is a mathematical model that exists in the nervous system of a computer. It is connected by a large number of neurons and performs calculations. It changes the internal structure based on external information and is often used to process input. Model complex relationships between outputs. The structure of a basic neural network has an input layer, a hidden layer, and an output layer. The picture below is a neural network diagram,



2. Input layer: receives sensory information;
3. Hidden layer: processing of input information;

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

2.3.4. Build and train 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 the 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, generally including the following three:

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

  Please refer to the official documentation: [tf.keras.layers.Flatten | TensorFlow Core v2.8.0 (google.cn)]

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

  Please refer to the official documentation: [tf.keras.layers.Dense | TensorFlow Core v2.8.0 (google.cn)]

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

  Please refer to the official documentation: [tf.keras.layers.Conv2D | TensorFlow Core v2.8.0 (google.cn)]

3. Configure the training method for training the neural network

```
model.compile( optimizer = optimizer, loss = loss function, metrics =
["accuracy"])
```

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=label of the training set,
batch_size=specifies the number of samples included in each batch when performing
gradient descent,
epochs = value at the end of training, validation_data = (input features of the
test set, label of the test set),
validation_split = what proportion is divided from the training set to the test
set,
validation_freq = number of epoch intervals for testing)
```

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 example of training cat and dog images

1), code path reference

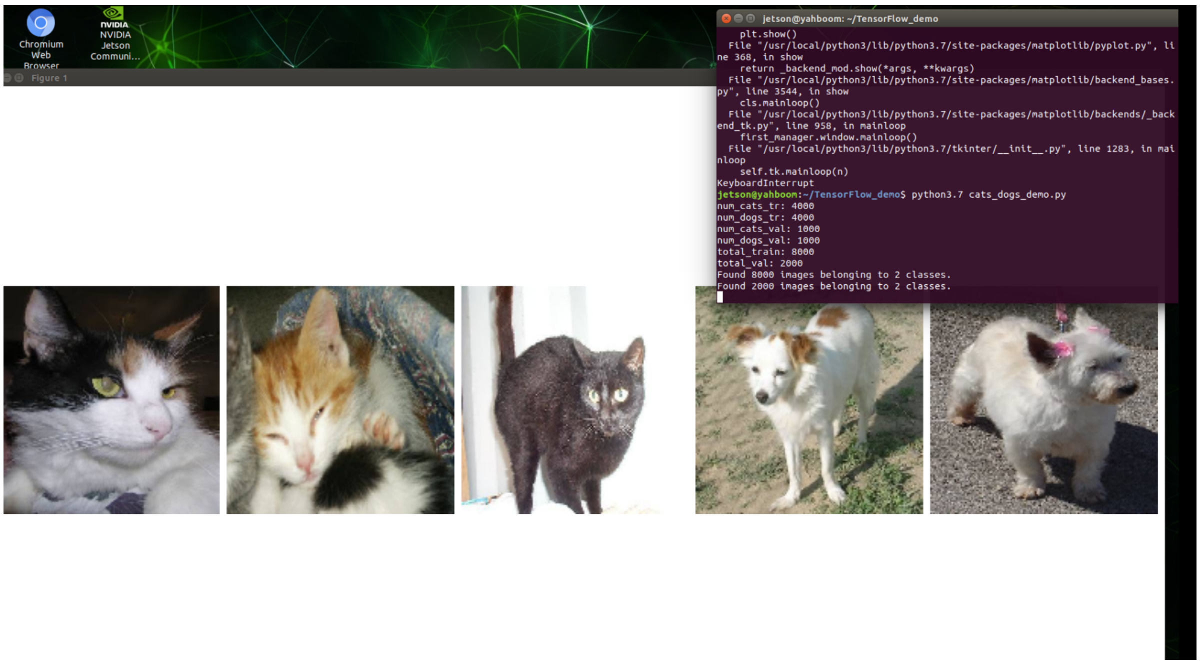
```
~/TensorFlow_demo/cats_dogs_demo.py
```

2), run the program

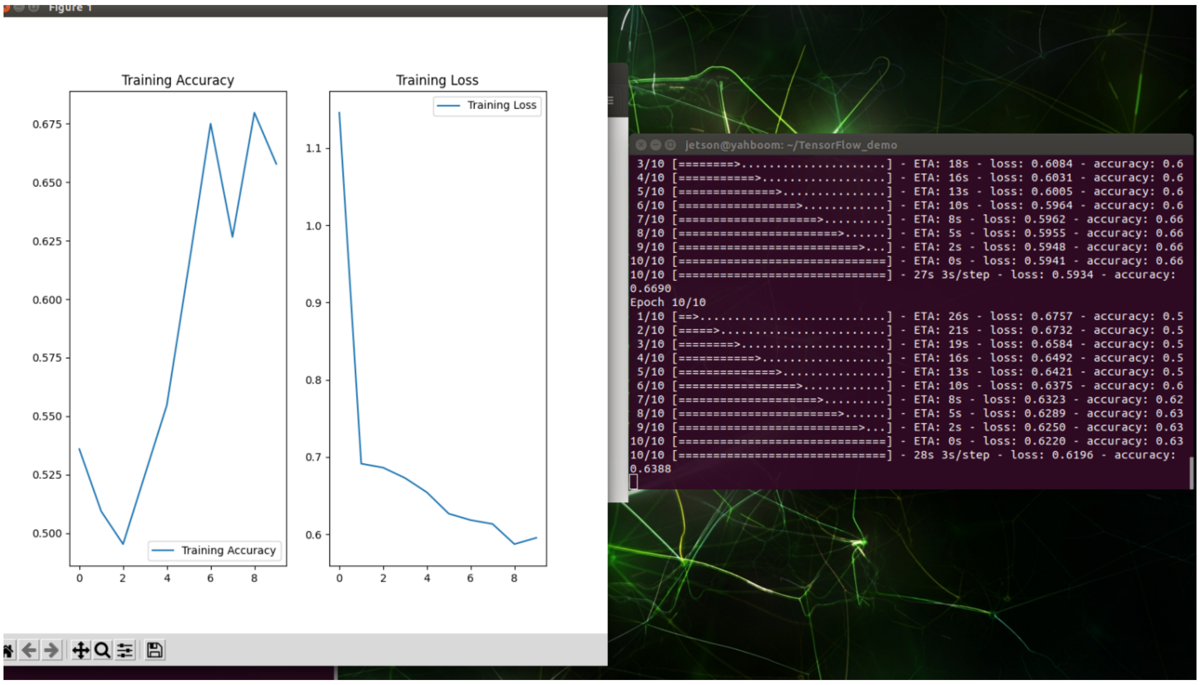
```
cd TensorFlow_demo/
python3 cats_dogs_demo.py
```

3. Screenshots of program running

When pictures of kittens and puppies appear, in the picture display window, press the q key to continue executing the program.



```
jetson@yahboom: ~/TensorFlow_demo
    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.
```



```
jetson@yahboom: ~/TensorFlow_demo
 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 number of training epochs for this model is 10, and the number of sample batches is 10. The coordinate curve on the left shows that as the number of training times increases, the accuracy acc and error loss will increase and decrease.