2.Basic use of TensorFlow

2.Basic use of TensorFlow
- 2.1 What is TensorFlow
  - 2.1.1 Definition
  - 2.1.2 core components
- 2.2 TensorFlow 2
  - 2.2.1 Introduction
  - 2.2.2 Upgrading direction
- 2.3 TensorFlow basic concept syntax
  - 2.3.1 Tensor
  - 2.3.2 Use the Eager Execution(dynamic graph) mechanism to operate on tensors
  - 2.3.3 Neural network
  - 2.3.4 Building and training neural network(**kear**)
  - 2.3.5 Example of training neural network - classic example of training cat and dog images

2.1 What is TensorFlow

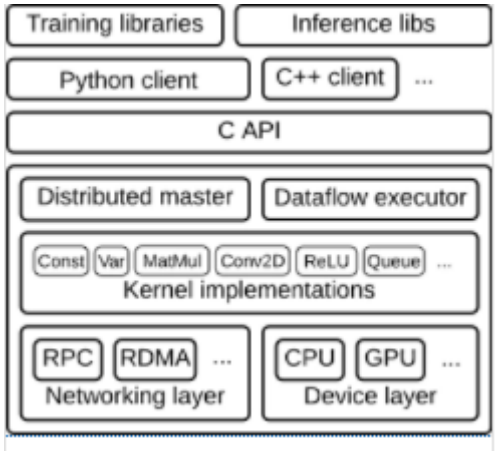
2.1.1 Definition

TensorFlow™ is a dataflow programming, which is widely used in the programming implementation of various machine learning algorithms. Its predecessor was Google 's neural 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 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: distribution center(distributed master), executor(dataflow executor/worker service), kernel application(kernel implementation), and the bottommost device layer (device layer)/ network layer (layer).



1. distribution center(distributed master)

The distribution center clips the subgraph from the input data flow graph, divides it into operation fragments and starts the executor. The distribution center will perform preset operation optimizations when processing the data flow graph, 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. Executors choose to perform [parallel computation]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 computation of tensors, cuDNN library etc. to perform GPU acceleration, gemmlowp to perform low numerical precision computation, and users can register additional kernels(fused kernels) in the kernel application to improve basic operations, such as excitation functions and The running efficiency of its gradient calculation.

2.2 TensorFlow 2

2.2.1 Introduction

TensorFlow is an open source tool for deep learning 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 Upgrading direction

1. Easily build models using Keras and Eager Execution.
2. to achieve a robust production environment model deployment on any platform.
3. to provide powerful experimental tools for research.
4. Simplify the API by cleaning up deprecated 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 properties: data type(default: float32) and shape. The data types are roughly as shown in the table below,

Tensor type	describe
tf.float32	32-bit floating point number
tf.float64	64-bit floating point
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 two main types of tensor in TensorFlow, namely

- tf.Variable : Variable Tensor, which needs to specify the initial value. It is often used to define variable parameters, such as the weight of a neural network.
- tf.constant : constant Tensor, you need to specify the initial value, define a tensor that does not change

4. define a variable Tensor

Create a new python file, name it Tensor_Variable, and give it permission to execute,
(Note: The Python version used on the Raspberry Pi is 3.7. Please use python3.7 when running commands!)

(Note: You do not need to enter docker when running Raspberry Pi 5. Just open a new terminal to run. Please use python3 when running commands)

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

Paste the following code inside,

```
import  tensorflow  as  tf
v = tf.Variable([[ 1, 2 ], [ 3, 4 ]])     # 2D variable of shape(2, 2)
print(v)
```

run the test,

```
python3 Tensor_Variable.py
```

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 give it permission to execute,

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

Paste the following code inside,

```
import  tensorflow  as  tf
v = tf.constant([[ 1, 2 ], [ 3, 4 ]])     # 2D variable of 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), the commonly used method of creating a new special constant tensor:

- 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: Referring to a certain shape, create a new constant Tensor with all 1s

For example: v = tf.ones_like(c) # A constant Tensor that is consistent with the 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 a scalar value

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

This part of the code can be referred 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 of TensorFlow2 and TensorFlow1.x

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

2. take tensor addition as an example to illustrate,

Create a new python file, name it tensor_plus, and give it permission to execute,

```
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 python execution process is the same, and if it is in Tensorflow1.x, a session needs to be established, and the session performs the addition operation inside.

3. TensorFlow commonly used API:

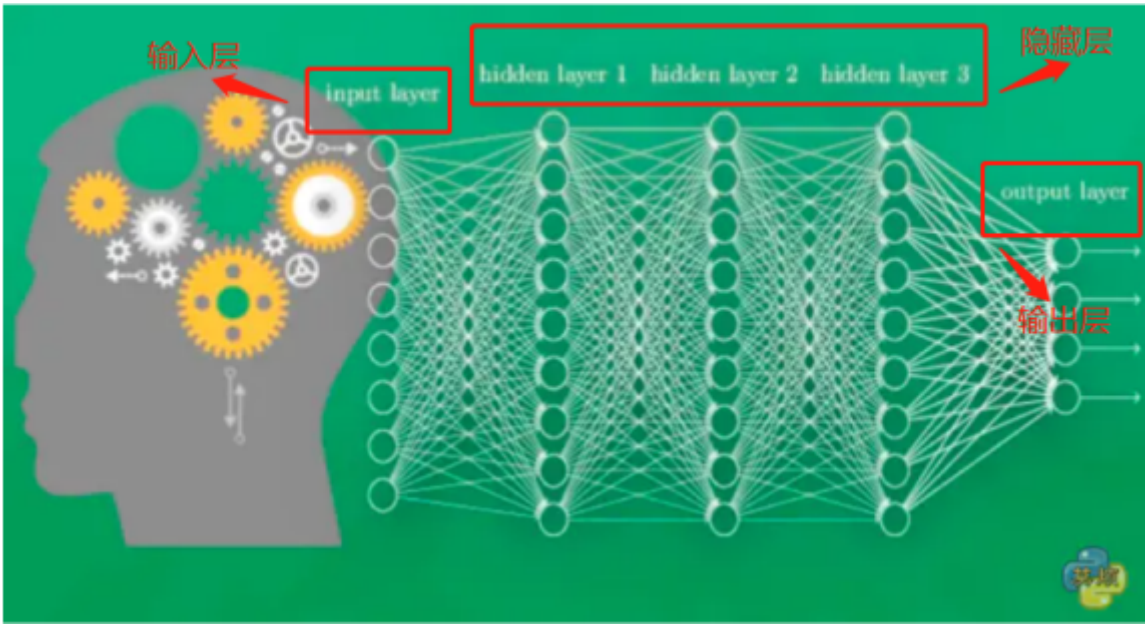
- tf.math: Mathematical calculation module, which provides a large number of mathematical calculation functions.
- tf.linalg: Linear algebra module, provides a large number of linear algebra calculation methods and classes.
- tf.image: Image processing module, provides classes like image cropping, transformation, encoding, decoding, etc.
- tf.train: Provides components for training, such as optimizers, learning rate decay strategies, etc.
- tf.nn: Provides the 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 high-level neural network layers from the original tf.layers.
- tf.data: Input data processing module, provides classes like tf.data.Dataset to encapsulate input data, specify batch size, etc.

For the 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, which exists in the nervous system of the computer. It is connected and calculated by a large number of neurons. On the basis of external information, the internal structure is changed. relationship is modeled. The structure of a basic neural network has an input layer, a hidden layer, and an output layer. The following figure is a neural network diagram,



2. input layer: receive perception information;
3. Hidden layer: processing the input information;
4. the output layer: the output computer's cognition of the input information.

2.3.4 Building and training neural network(kear)

1. import data data

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

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

2. define a structural network describing 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 there are the following three:

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

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

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

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

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

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 parameter is 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.

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 training set, y=label of training set,
batch_size=specify the number of samples included in each batch when gradient
descent is performed,
        eTOChs=value at the end of training,validation_data=(input features of
test set, labels of test set),
        validataion_split = how much to split from the training set to the test
set,
        validation_freq = number of eTOCh intervals for testing)
```

Example:

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

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()
```

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

2.3.5 Example of training neural network - classic example of training cat and dog images

1. code path reference

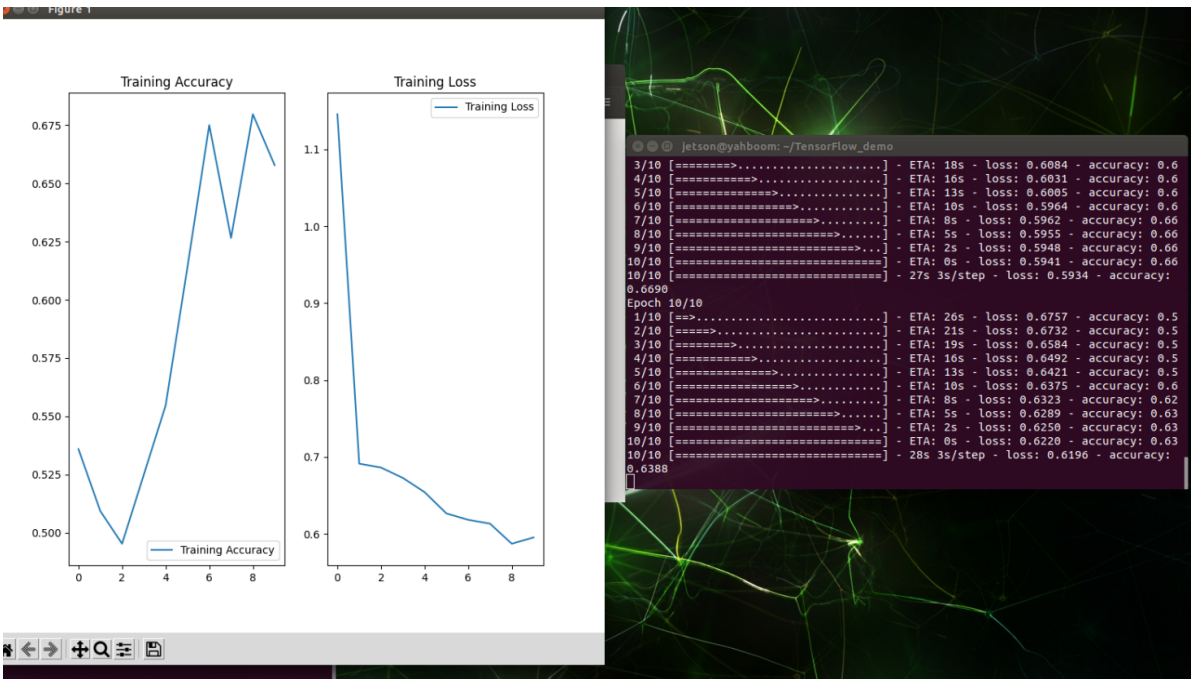
```
~/TensorFlow_demo/cats_dogs_demo.py
```

2. run the program

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

3. Screenshot of program running

When a cat or puppy photo appears, in the picture display window, press the q key to continue the program.



The number of training eTOChs for the model is 10, and the sample batch is 10. The coordinate curve on the left shows that as the number of training increases, both the accuracy acc and error loss rise and fall.