In [1]:
```python
import tensorflow as tf
```

In [2]:
```python
# 0D张量只包含一个数字，有0个维度，又称为标量
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)
```

```
tf.Tensor(4, shape=(), dtype=int32)
```

In [3]:
```python
# 1D张量包含一个一维数组，可以看作由0D张量组成的数组，有1个维度，又称为向量
rank_1_tensor = tf.constant([2.0,3.0,4.0])
print(rank_1_tensor)
```

```
tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)
```

In [4]:
```python
# 2D张量可以看作由1D张量组成的数组，有2个维度，又称为矩阵
rank_2_tensor = tf.constant([[1,2],
                             [3,4],
                             [5,6]], dtype=tf.float16)

print(rank_2_tensor)
```

```
tf.Tensor(
[[1. 2.]
 [3. 4.]
 [5. 6.]], shape=(3, 2), dtype=float16)
```

In [5]:
```python
# 3D张量可以看作由2D张量组成的数组，有3个维度

rank_3_tensor = tf.constant([
    [[0,1,2,3,4],
    [5,6,7,8,9]],
    [[10,11,12,13,14],
    [15,16,17,18,19]],
    [[20,21,22,23,24],
    [25,26,27,28,29]],
])

print(rank_3_tensor)
```

```
tf.Tensor(
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]

 [[20 21 22 23 24]
  [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32)
```

1·标量运算

·向量运算
·矩阵运算

In [6]:
```python
#标量运算:对张量实施逐元素运算，包括加、减、乘、除、乘方以及三角函数、指数、对数等常见

a = tf.constant([[1,2],
                 [3,4]])
```

```
b = tf.constant([[0,0],
                 [1,0]])

print(a + b)
print(a - b)
print(a * b)
print(a / b)
```

```
tf.Tensor(
[[1 2]
 [4 4]], shape=(2, 2), dtype=int32)
tf.Tensor(
[[1 2]
 [2 4]], shape=(2, 2), dtype=int32)
tf.Tensor(
[[0 0]
 [3 0]], shape=(2, 2), dtype=int32)
tf.Tensor(
[[inf inf]
 [ 3.  inf]], shape=(2, 2), dtype=float64)
```

1.标量运算

2.向量运算

3.矩阵运算

In [7]:

```
# 向量运算:只在一个特定轴上运算，将一个向量映射到一个标量或者另外一个向量
A= tf.constant([[2,20,30,3,6],
                 [1,1,1,1,1]])

print(tf.math.reduce_sum(A))
print(tf.math.reduce_max(A))

B = tf.constant([[2,20,30,3,6],
                 [3,11,16,1,8],
                 [14,45,23,5,27]])

print(tf.math.reduce_sum(B,0))#沿0轴（列方向），求和
print(tf.math.reduce_sum(B,1))#沿1轴(行方向)，求和
print(tf.math.reduce_max(B,0))# 沿0轴，求最大值
print(tf.math.reduce_max(B,1))# 沿1轴，求最大值
```

```
tf.Tensor(66, shape=(), dtype=int32)
tf.Tensor(30, shape=(), dtype=int32)
tf.Tensor([19 76 69  9 41], shape=(5,), dtype=int32)
tf.Tensor([ 61  39 114], shape=(3,), dtype=int32)
tf.Tensor([14 45 30  5 27], shape=(5,), dtype=int32)
tf.Tensor([30 16 45], shape=(3,), dtype=int32)
```

In [8]:

```
# 矩阵运算:矩阵必须是二维的，包括矩阵乘法、矩阵转置、矩阵逆、矩阵行列式、矩阵求特征值、
A= tf.constant([[2,20,30,3,6],
                 [1,1,1,1,1]])

#矩阵转置
A_trans = tf.linalg.matrix_transpose(A)
print(A_trans)

B= tf.constant([[2,20,30,3,6],
                 [3,11,16,1,8],
                 [14,45,23,5,27]])
#矩阵点积
print(tf.linalg.matmul(B, A_trans))
```

```
tf.Tensor(
[[ 2  1]
 [20  1]
 [30  1]
 [ 3  1]
 [ 6  1]], shape=(5, 2), dtype=int32)
tf.Tensor(
[[1349   61]
 [ 757   39]
 [1795  114]], shape=(3, 2), dtype=int32)
```

2、低阶API

. tf.constant():提供了常量的声明功能

. tf.Variable():提供了变量的声明功能

. tf.reshape():提供了多阶Tensor的形状变换功能

. tf.math.reduce_mean():提供了对Tensor求平均值的功能

. tf.random.normal():随机生成一个Tensor，其值符合正态分布

. tf.random.uniform():随机生成一个Tensor，其值符合均匀分布

. tf.transpose():提供了矩阵的转置功能

. tf.math.argmax():提供了返回一个数组内最大值对应索引的功能

. tf.expand_dims():在输入的Tensor中增加一个维度

. tf.concat():将多个Tensor在同一个维度上进行连接

. tf.bitcast():提供了数据类型转换功能

In [9]:

```
# tf.constant()提供了常量的声明功能
a=tf.constant(7)
print(a)
print(a.numpy())
```

```
tf.Tensor(7, shape=(), dtype=int32)
7
```

In [16]:

```
# tf.Variable():提供了变量的声明功能

#声明一个Python变量
a1 = 7
print('a1= ',a1)
#声明一个0阶Tensor变量
a2 = tf.Variable(7)
print('a2= ',a2)
#声明一个1阶Tensor变量，即数组
a3 = tf.Variable([0,1,2])
print('a3= ',a3)
print(a1,a2,a3)
```

```
a1=  7
a2=  <tf.Variable 'Variable:0' shape=() dtype=int32, numpy=7>
a3=  <tf.Variable 'Variable:0' shape=(3,) dtype=int32, numpy=array([0, 1, 2])> /n
7 <tf.Variable 'Variable:0' shape=() dtype=int32, numpy=7> <tf.Variable 'Variable:0' s
hape=(3,) dtype=int32, numpy=array([0, 1, 2])>
```

In [17]:

```
# tf.reshape():提供了多阶Tensor的形状变换功能

a = tf.Variable([[0,1,2],[3,4,5]])
print(a)
# 对a的形状进行变换，变换为(3,2)
a1 = tf.reshape(a,[3,2])
```

```
print(a1)
print(a1.shape)
```

```
<tf.Variable 'Variable:0' shape=(2, 3) dtype=int32, numpy=
array([[0, 1, 2],
       [3, 4, 5]])>
tf.Tensor(
[[0 1]
 [2 3]
 [4 5]], shape=(3, 2), dtype=int32)
(3, 2)
```

In [18]:

```
# t.math.reduce_mean():提供了对Tensor求平均值的功能，输出数据类型会根据输入数据类型来
# input_tensor:配置输入的Tensor
# axis:配置按行求平均值或按列求平均值，默认是全行全列求平均值
# keepdims:配置输出结果是否保持二维矩阵特性
# name:配置操作的名称

a = tf.constant([1,2.,3,4,5,6,7.])

#输入数据类型是float32，输出数据类型也是float32
print(a.dtype)
print(tf.math.reduce_mean(a))
b = tf.constant([[1,2,1],[5,2,10]])

#输入数据类型是int32，输出数据类型也是int32
print(b.dtype)

#虽然平均值为3.5，但是由于上面确定了输出类型为整型，因此强制赋值为整数3
print(tf.math.reduce_mean(b))
```

```
<dtype: 'float32'>
tf.Tensor(4.0, shape=(), dtype=float32)
<dtype: 'int32'>
tf.Tensor(3, shape=(), dtype=int32)
```

In [20]:

```
# tf.random.normal():随机生成一个Tensor，其值符合正态分布。使用该API时有如下参数需要配
# shape:配置生成Tensor的维度
# mean:配置正态分布的中心值
# stddev:配置正态分布的标准差
# seed:配置正态分布的随机生成粒子
# dtype:配置生成Tensor的数据类型

a = tf.random.normal(shape=[2,3], mean=2)
print(a)# a为Tensor类型，是一个2维张量Tensor
print(type(a))# type()方法是Python的原生方法，查看Tensor对象a的类型，是一个Tensor
print(a.dtype)#由于a是Tensor类型，因此可以使用Tensor对象的dtype属性
print(a.numpy())# a转换成numpy的array数组类型，就是普通的数组类型
print(type(a.numpy()))
```

```
tf.Tensor(
[[3.093015   0.83826673 0.6251465 ]
 [3.308845   3.5064597  2.4058805 ]], shape=(2, 3), dtype=float32)
<class 'tensorflow.python.framework.ops.EagerTensor'>
<dtype: 'float32'>
[[3.093015   0.83826673 0.6251465 ]
 [3.308845   3.5064597  2.4058805 ]]
<class 'numpy.ndarray'>
```

In [24]:

```
# tf.random.uniform():随机生成一个Tensor，其值符合均匀分布。有如下参数需要配置:
# shape:配置生成Tensor的维度
# minval: 配置随机生成数值的最小值
# maxval:配置随机生成数值的最大值
# seed:配置正态分布的随机生成粒子
```

```
# dtype:配置生成Tensor的数据类型

a = tf.random.uniform(shape=[2,3],minval=1,maxval=10,seed=8,dtype=tf.int32)
print(a.numpy)
print('*'*20)
print(a.numpy())
```

```
<bound method _EagerTensorBase.numpy of <tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[4, 1, 4],
       [2, 5, 9]])>>
********************
[[4 1 4]
 [2 5 9]]
```

In [25]:

```
# tf.transpose():提供了矩阵的转置功能。使用该API时配置的参数如下:
# a:输入需要转置的矩阵
# perm:配置转置后矩阵的形状
# conjugate:当输入矩阵是复数时，需要配置为
# Truename:配置本次操作的名称

#定义x为一个3维张量，形状是(2,2,3)
x = tf.constant([[[1,2,3],
                  [4,5,6]],
                 [[7,8,9],
                  [10,11,12]]])
#转置后的形状是(2,3,2)
a = tf.transpose(x,perm=[0,2,1])
print(a.numpy())
```

```
[[[ 1  4]
  [ 2  5]
  [ 3  6]]

 [[ 7 10]
  [ 8 11]
  [ 9 12]]]
```

In [26]:

```
# tf.math.argmax():提供了返回一个数组内最大值对应索引的功能。使用该API时有如下参数可以
# input: 配置输入的数组
# axis:配置计算的维度
# output_type:配置输出的格式
# name:配置操作的名称

a = tf.constant([1,2,3,4,5])
x = tf.math.argmax(a)
print(x.numpy())
```

```
4
```

In [27]:

```
# tf.expand_dims():在输入的Tensor中增加一个维度，比如t是一个维度为[2]的Tensor，那么tf.
# 使用这个AP1I时需要配置如下参数:
# input:配置输入的Tensor
# axis:配置需要添加维度的下标，比如[2，1]需要在2和1之间添加，则配置值为1
# name:配置输出Tensor的名称

#初始化一个维度为(3,1)的Tensor
a = tf.constant([[1],[2],[3]])
print(a.shape)
print(a)
# 为a增加一个维度，使其维度变成(1,3,1)
b = tf.expand_dims(a,0)
print(b.shape)
print(b)
```

```
(3, 1)
tf.Tensor(
[[1]
 [2]
 [3]], shape=(3, 1), dtype=int32)
(1, 3, 1)
tf.Tensor(
[[[1]
  [2]
  [3]]], shape=(1, 3, 1), dtype=int32)
```

In [33]:

```
# tf.concat():将多个Tensor在同一个维度上进行连接，使用该API时需要进行如下参数配置:
# values:配置Tensor的列表或者是一个单独的Tensor
# axis:配置按行或按列连接,axis=0表示按行连接，axis=1表示按列连接
# name:配置运算操作的名称

a1 = tf.constant([[2,3,4],[4,5,6],[2,3,4]])
a2 = tf.constant([[1,2,2],[6,7,9],[2,3,2]])
print('原始矩阵：')
print('a1=',a1.numpy())
print('a2=',a2.numpy())
#按行进行连接
print('按行进行连接:axis=0')
b = tf.concat([a1,a2], axis=0)
print(b.numpy())

#按列进行连接
print('按列进行连接:axis=1')
b = tf.concat([a1,a2], axis=1)
print(b.numpy())
```

```
原始矩阵：
a1= [[2 3 4]
 [4 5 6]
 [2 3 4]]
a2= [[1 2 2]
 [6 7 9]
 [2 3 2]]
按行进行连接:axis=0
[[2 3 4]
 [4 5 6]
 [2 3 4]
 [1 2 2]
 [6 7 9]
 [2 3 2]]
按列进行连接:axis=1
[[2 3 4 1 2 2]
 [4 5 6 6 7 9]
 [2 3 4 2 3 2]]
```

In [34]:

```
# tf.bitcast():提供了数据类型转换功能:
# type:配置转换后的数据类型，可选择的类型包括：
# tf.bfloat16, tf.half, tf.float32, tf.float64 (加tf.+类型)

#a原本是浮点类型float32类型
a = tf.constant(32.0)
#将a转换成整型int32
b = tf.bitcast(a,type=tf.int32)
print(a.dtype)
print(b.dtype)
```

```
<dtype: 'float32'>
<dtype: 'int32'>
```

3、Tensorflow高阶API(tf.keras)

In [47]:

```
from sklearn import datasets
import numpy as np

#从sklearn中导入数据集
x_train = datasets.load_iris().data #导入iris数据集的输入
y_train = datasets.load_iris().target #导入iris数据集的标签

np.random.seed(120) #设置随机种子，让每次结果都一样，方便对照
np.random.shuffle(x_train) #使用shuffle()方法，让输入x_train乱序
np.random.seed(120) #设置随机种子，让每次结果都一样，方便对照
np.random.shuffle(y_train) #使用shuffle()方法，让输入y_train乱序

#tf.random.set_seed(120) #让tensorflow中的种子数设置为120'''
```

In [48]:

```
# 模型创建(Sequential方法):实例化模型对象，
# 方法是: tf.keras.Sequential()
import tensorflow as tf
#创建模型对象的实例
#model = tf.keras.Sequential()
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(3, activation='softmax',
                          kernel_regularizer=tf.keras.regularizers.l2())
])
```

In [49]:

```
'''# 模型创建(Sequential方法):组建网络层

# 示例代码:实现三个全连接神经网络层级的集成，构建一个全连接神经网络模型
import tensorflow as tf
#使用add()方法集成神经网络层级
model.add(tf.keras.layers.Dense(256, activation="relu"))
model.add(tf.keras.layers.Dense(128,activation= "relu"))
model.add(tf.keras.layers.Dense(2,activation="softmax"))'''
```

Out[49]:

```
'# 模型创建(Sequential方法):组建网络层\n\n# 示例代码:实现三个全连接神经网络层级的集成，构建一个全连接神经网络模型\nimport tensorflow as tf\n#使用add()方法集成神经网络层级\nmodel.add(tf.keras.layers.Dense(256, activation="relu"))\nmodel.add(tf.keras.layers.Dense(128,activation= "relu"))\nmodel.add(tf.keras.layers.Dense(2,activation="softmax"))'
```

Sequential().compile():提供了神经网络模型的编译功能，需要定义三个参数:

> loss:用来配置模型的损失函数，可以通过名称调用tf.losses （API中已经定义好的loss函数)
> optimizer:用来配置模型的优化器，可以调用tf.keras.optimizers （API配置模型所需要的优化器)
> metrics:用来配置模型评价的方法，如accuracy、mse等

In [50]:

```
# 编译模型
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=["sparse_categorical_accuracy"])
```

```
C:\Users\A\anaconda3\lib\site-packages\keras\optimizer_v2\gradient_descent.py:102: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

In [51]:

```
# 模型训练
```

```python
#使用model.fit()方法来执行训练过程,
model.fit(
    x_train, y_train,
    batch_size=32,
    epochs=100,
    validation_split=0.2,
    validation_freq=20)
```

```
Epoch 1/100
4/4 [==============================] - 0s 2ms/step - loss: 1.0306 - sparse_categorical_accuracy: 0.5417
Epoch 2/100
4/4 [==============================] - 0s 2ms/step - loss: 0.9619 - sparse_categorical_accuracy: 0.6083
Epoch 3/100
4/4 [==============================] - 0s 3ms/step - loss: 1.0652 - sparse_categorical_accuracy: 0.6583
Epoch 4/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7098 - sparse_categorical_accuracy: 0.6833
Epoch 5/100
4/4 [==============================] - 0s 3ms/step - loss: 0.8163 - sparse_categorical_accuracy: 0.6583
Epoch 6/100
4/4 [==============================] - 0s 3ms/step - loss: 1.0491 - sparse_categorical_accuracy: 0.6500
Epoch 7/100
4/4 [==============================] - 0s 2ms/step - loss: 0.8715 - sparse_categorical_accuracy: 0.6333
Epoch 8/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7782 - sparse_categorical_accuracy: 0.7000
Epoch 9/100
4/4 [==============================] - 0s 4ms/step - loss: 1.2221 - sparse_categorical_accuracy: 0.6000
Epoch 10/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5530 - sparse_categorical_accuracy: 0.7667
Epoch 11/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5949 - sparse_categorical_accuracy: 0.7083
Epoch 12/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5465 - sparse_categorical_accuracy: 0.7917
Epoch 13/100
4/4 [==============================] - 0s 7ms/step - loss: 0.7197 - sparse_categorical_accuracy: 0.6500
Epoch 14/100
4/4 [==============================] - 0s 4ms/step - loss: 0.8596 - sparse_categorical_accuracy: 0.6167
Epoch 15/100
4/4 [==============================] - 0s 10ms/step - loss: 0.6518 - sparse_categorical_accuracy: 0.7667
Epoch 16/100
4/4 [==============================] - 0s 3ms/step - loss: 0.8912 - sparse_categorical_accuracy: 0.6333
Epoch 17/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5538 - sparse_categorical_accuracy: 0.8167
Epoch 18/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5209 - sparse_categorical_accuracy: 0.7833
Epoch 19/100
4/4 [==============================] - 0s 3ms/step - loss: 0.6197 - sparse_categorical_accuracy: 0.6833
Epoch 20/100
4/4 [==============================] - 0s 85ms/step - loss: 0.5462 - sparse_categorical_accuracy: 0.7167 - val_loss: 0.4783 - val_sparse_categorical_accuracy: 0.8333
Epoch 21/100
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.4707 - sparse_categorical
_accuracy: 0.8333
Epoch 22/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5747 - sparse_categorical
_accuracy: 0.7500
Epoch 23/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5052 - sparse_categorical
_accuracy: 0.7417
Epoch 24/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5326 - sparse_categorical
_accuracy: 0.8333
Epoch 25/100
4/4 [==============================] - 0s 3ms/step - loss: 0.6479 - sparse_categorical
_accuracy: 0.8333
Epoch 26/100
4/4 [==============================] - 0s 4ms/step - loss: 0.4549 - sparse_categorical
_accuracy: 0.9083
Epoch 27/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4777 - sparse_categorical
_accuracy: 0.8667
Epoch 28/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4313 - sparse_categorical
_accuracy: 0.9083
Epoch 29/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5341 - sparse_categorical
_accuracy: 0.7583
Epoch 30/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5056 - sparse_categorical
_accuracy: 0.8250
Epoch 31/100
4/4 [==============================] - 0s 4ms/step - loss: 0.5137 - sparse_categorical
_accuracy: 0.7833
Epoch 32/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4074 - sparse_categorical
_accuracy: 0.8667
Epoch 33/100
4/4 [==============================] - 0s 4ms/step - loss: 0.4537 - sparse_categorical
_accuracy: 0.8083
Epoch 34/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4198 - sparse_categorical
_accuracy: 0.9167
Epoch 35/100
4/4 [==============================] - 0s 2ms/step - loss: 0.6320 - sparse_categorical
_accuracy: 0.6333
Epoch 36/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5572 - sparse_categorical
_accuracy: 0.7750
Epoch 37/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5676 - sparse_categorical
_accuracy: 0.7167
Epoch 38/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5682 - sparse_categorical
_accuracy: 0.7333
Epoch 39/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5107 - sparse_categorical
_accuracy: 0.7167
Epoch 40/100
4/4 [==============================] - 0s 12ms/step - loss: 0.5591 - sparse_categorica
l_accuracy: 0.7500 - val_loss: 0.6830 - val_sparse_categorical_accuracy: 0.7000
Epoch 41/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5818 - sparse_categorical
_accuracy: 0.8000
Epoch 42/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3993 - sparse_categorical
_accuracy: 0.8667
Epoch 43/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4115 - sparse_categorical
_accuracy: 0.9083
Epoch 44/100
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.4853 - sparse_categorical
_accuracy: 0.7750
Epoch 45/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4163 - sparse_categorical
_accuracy: 0.8833
Epoch 46/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4727 - sparse_categorical
_accuracy: 0.8000
Epoch 47/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4674 - sparse_categorical
_accuracy: 0.8167
Epoch 48/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4536 - sparse_categorical
_accuracy: 0.8917
Epoch 49/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4905 - sparse_categorical
_accuracy: 0.9000
Epoch 50/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4394 - sparse_categorical
_accuracy: 0.8250
Epoch 51/100
4/4 [==============================] - 0s 2ms/step - loss: 0.6971 - sparse_categorical
_accuracy: 0.6333
Epoch 52/100
4/4 [==============================] - 0s 4ms/step - loss: 0.5546 - sparse_categorical
_accuracy: 0.8000
Epoch 53/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4602 - sparse_categorical
_accuracy: 0.7750
Epoch 54/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5039 - sparse_categorical
_accuracy: 0.7833
Epoch 55/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3859 - sparse_categorical
_accuracy: 0.9250
Epoch 56/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3940 - sparse_categorical
_accuracy: 0.9083
Epoch 57/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3844 - sparse_categorical
_accuracy: 0.9417
Epoch 58/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4611 - sparse_categorical
_accuracy: 0.8250
Epoch 59/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4397 - sparse_categorical
_accuracy: 0.8667
Epoch 60/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4033 - sparse_categorica
l_accuracy: 0.9167 - val_loss: 0.3977 - val_sparse_categorical_accuracy: 0.9667
Epoch 61/100
4/4 [==============================] - 0s 4ms/step - loss: 0.4098 - sparse_categorical
_accuracy: 0.8667
Epoch 62/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4387 - sparse_categorical
_accuracy: 0.8333
Epoch 63/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4040 - sparse_categorical
_accuracy: 0.8667
Epoch 64/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5888 - sparse_categorical
_accuracy: 0.8250
Epoch 65/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3751 - sparse_categorical
_accuracy: 0.9500
Epoch 66/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3872 - sparse_categorical
_accuracy: 0.9167
Epoch 67/100
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.5178 - sparse_categorical
_accuracy: 0.7417
Epoch 68/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4354 - sparse_categorical
_accuracy: 0.8333
Epoch 69/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3686 - sparse_categorical
_accuracy: 0.9417
Epoch 70/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3792 - sparse_categorical
_accuracy: 0.9250
Epoch 71/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4358 - sparse_categorical
_accuracy: 0.8333
Epoch 72/100
4/4 [==============================] - 0s 4ms/step - loss: 0.4314 - sparse_categorical
_accuracy: 0.8583
Epoch 73/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3735 - sparse_categorical
_accuracy: 0.9500
Epoch 74/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4658 - sparse_categorical
_accuracy: 0.8000
Epoch 75/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5149 - sparse_categorical
_accuracy: 0.7333
Epoch 76/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4693 - sparse_categorical
_accuracy: 0.8250
Epoch 77/100
4/4 [==============================] - 0s 3ms/step - loss: 0.5652 - sparse_categorical
_accuracy: 0.7250
Epoch 78/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3974 - sparse_categorical
_accuracy: 0.9167
Epoch 79/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3948 - sparse_categorical
_accuracy: 0.9333
Epoch 80/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3998 - sparse_categorica
l_accuracy: 0.8917 - val_loss: 0.4344 - val_sparse_categorical_accuracy: 0.8000
Epoch 81/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3705 - sparse_categorical
_accuracy: 0.9417
Epoch 82/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4268 - sparse_categorical
_accuracy: 0.9083
Epoch 83/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3839 - sparse_categorical
_accuracy: 0.9500
Epoch 84/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4830 - sparse_categorical
_accuracy: 0.8167
Epoch 85/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4201 - sparse_categorical
_accuracy: 0.9000
Epoch 86/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4669 - sparse_categorical
_accuracy: 0.8250
Epoch 87/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5585 - sparse_categorical
_accuracy: 0.7917
Epoch 88/100
4/4 [==============================] - 0s 3ms/step - loss: 0.4020 - sparse_categorical
_accuracy: 0.8917
Epoch 89/100
4/4 [==============================] - 0s 2ms/step - loss: 0.5704 - sparse_categorical
_accuracy: 0.7167
Epoch 90/100
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.4474 - sparse_categorical
_accuracy: 0.8333
Epoch 91/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4752 - sparse_categorical
_accuracy: 0.8333
Epoch 92/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4194 - sparse_categorical
_accuracy: 0.8917
Epoch 93/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3423 - sparse_categorical
_accuracy: 0.9750
Epoch 94/100
4/4 [==============================] - 0s 4ms/step - loss: 0.3820 - sparse_categorical
_accuracy: 0.9333
Epoch 95/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3612 - sparse_categorical
_accuracy: 0.9500
Epoch 96/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3614 - sparse_categorical
_accuracy: 0.9750
Epoch 97/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4358 - sparse_categorical
_accuracy: 0.8583
Epoch 98/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3838 - sparse_categorical
_accuracy: 0.9250
Epoch 99/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3695 - sparse_categorical
_accuracy: 0.9417
Epoch 100/100
4/4 [==============================] - 0s 33ms/step - loss: 0.4284 - sparse_categorica
l_accuracy: 0.8833 - val_loss: 0.4003 - val_sparse_categorical_accuracy: 0.9000
```

Out[51]:
```
<keras.callbacks.History at 0x1e45a22d8e0>
```

In [52]:
```
# 打印出网络结构和参数统计
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 3)                 15

=================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________________________
```

In [54]:
```
#保存模型
model.save(filepath='demo_model')#文件夹(将模型保存此文件夹)
```

```
INFO:tensorflow:Assets written to: demo_model\assets
```

In [58]:
```
#模型加载
model = tf.keras.models.load_model("demo_model")
#模型预测
model.predict(x_train)
```

Out[58]:
```
array([[3.14646736e-02, 8.80347610e-01, 8.81876498e-02],
       [2.91265990e-03, 3.06028843e-01, 6.91058457e-01],
       [3.10491994e-02, 8.40576589e-01, 1.28374189e-01],
       [5.13052121e-02, 8.30297530e-01, 1.18397325e-01],
       [4.63178178e-04, 1.02751993e-01, 8.96784842e-01],
       [8.83611321e-01, 1.16224468e-01, 1.64192577e-04],
```

```
[3.10169882e-03, 3.65624517e-01, 6.31273806e-01],
[4.76708114e-02, 8.25890422e-01, 1.26438752e-01],
[5.54632656e-02, 7.46115804e-01, 1.98420912e-01],
[2.26356811e-03, 3.43816638e-01, 6.53919816e-01],
[9.48132694e-01, 5.18323556e-02, 3.49386755e-05],
[6.60215318e-02, 8.42074394e-01, 9.19040591e-02],
[9.40329611e-01, 5.96287921e-02, 4.15686736e-05],
[5.14234304e-02, 8.33117962e-01, 1.15458585e-01],
[1.81670357e-02, 8.43534648e-01, 1.38298303e-01],
[9.17307794e-01, 8.26356485e-02, 5.65361624e-05],
[9.39771831e-01, 6.02021515e-02, 2.59403951e-05],
[8.73562276e-01, 1.26236737e-01, 2.01004674e-04],
[9.68790174e-01, 3.11986431e-02, 1.12356911e-05],
[9.25321579e-01, 7.46142641e-02, 6.41916777e-05],
[1.22371421e-03, 2.48374701e-01, 7.50401556e-01],
[9.54921126e-01, 4.50480804e-02, 3.07518349e-05],
[4.10698168e-03, 3.43781412e-01, 6.52111590e-01],
[9.45350707e-01, 5.46112806e-02, 3.81244718e-05],
[2.77372915e-02, 7.40720153e-01, 2.31542572e-01],
[2.88150017e-03, 4.11123037e-01, 5.85995495e-01],
[1.25190064e-01, 8.03570688e-01, 7.12391883e-02],
[9.01653290e-01, 9.82772633e-02, 6.94702685e-05],
[9.73266482e-01, 2.67236233e-02, 9.84711187e-06],
[7.59080470e-01, 2.40259469e-01, 6.60091348e-04],
[8.89172137e-01, 1.10638991e-01, 1.88884005e-04],
[9.10034716e-01, 8.98097083e-02, 1.55573027e-04],
[9.03159916e-01, 9.66712013e-02, 1.68949526e-04],
[5.77937717e-05, 1.24903783e-01, 8.75038385e-01],
[3.80941643e-03, 4.69696850e-01, 5.26493728e-01],
[1.48676627e-03, 3.68532240e-01, 6.29980922e-01],
[4.89064977e-02, 8.36465776e-01, 1.14627831e-01],
[9.26056802e-01, 7.38291889e-02, 1.13946575e-04],
[2.91265990e-03, 3.06028843e-01, 6.91058457e-01],
[2.01539183e-03, 2.57866740e-01, 7.40117848e-01],
[9.34678972e-01, 6.52823895e-02, 3.87120526e-05],
[4.76962104e-02, 8.67050409e-01, 8.52533728e-02],
[2.10671895e-03, 3.29072714e-01, 6.68820560e-01],
[9.54432413e-04, 2.12133408e-01, 7.86912143e-01],
[8.89274478e-01, 1.10531338e-01, 1.94160559e-04],
[8.23044160e-04, 3.65772307e-01, 6.33404672e-01],
[2.71438668e-03, 4.99126464e-01, 4.98159111e-01],
[9.50373530e-01, 4.96026650e-02, 2.38452540e-05],
[3.88161745e-03, 2.78791785e-01, 7.17326581e-01],
[2.56814156e-02, 7.12943912e-01, 2.61374623e-01],
[2.16122661e-02, 7.36401975e-01, 2.41985768e-01],
[8.93695831e-01, 1.06192864e-01, 1.11329195e-04],
[9.50050235e-01, 4.99348380e-02, 1.49981961e-05],
[2.29931790e-02, 7.00599909e-01, 2.76406914e-01],
[5.20463521e-03, 4.71697003e-01, 5.23098350e-01],
[5.15393242e-02, 7.97611833e-01, 1.50848866e-01],
[2.84062922e-02, 6.48696482e-01, 3.22897255e-01],
[9.72117722e-01, 2.78666224e-02, 1.56845454e-05],
[6.31643683e-02, 8.00872624e-01, 1.35963038e-01],
[9.25618052e-01, 7.43047446e-02, 7.72165295e-05],
[8.55848014e-01, 1.43964544e-01, 1.87391313e-04],
[5.50287254e-02, 8.49210739e-01, 9.57605615e-02],
[5.14329113e-02, 7.16222465e-01, 2.32344642e-01],
[1.24799146e-03, 3.26811850e-01, 6.71940148e-01],
[1.19222666e-03, 2.51889139e-01, 7.46918619e-01],
[4.55351956e-02, 8.05210888e-01, 1.49253979e-01],
[1.09246455e-03, 4.07053411e-01, 5.91854155e-01],
[5.79300849e-03, 3.00974458e-01, 6.93232536e-01],
[9.50636685e-01, 4.93291765e-02, 3.42288258e-05],
[8.88542295e-01, 1.11338988e-01, 1.18760465e-04],
[4.53373678e-02, 8.70068073e-01, 8.45946148e-02],
[1.64195930e-03, 2.76643723e-01, 7.21714318e-01],
[3.79455741e-03, 4.44043696e-01, 5.52161753e-01],
[7.63758505e-03, 5.76052964e-01, 4.16309386e-01],
[9.78571236e-01, 2.14261152e-02, 2.69134398e-06],
```

```
[1.09009333e-01, 7.96622276e-01, 9.43684354e-02],
[5.28261214e-02, 8.76946032e-01, 7.02278689e-02],
[3.02484911e-02, 7.68602133e-01, 2.01149419e-01],
[8.73898804e-01, 1.25828072e-01, 2.73156271e-04],
[5.88886626e-03, 4.92140591e-01, 5.01970530e-01],
[1.83501747e-03, 3.62226874e-01, 6.35938108e-01],
[8.25725403e-03, 6.18111372e-01, 3.73631448e-01],
[1.04335072e-02, 5.61177433e-01, 4.28389043e-01],
[3.73378955e-03, 6.21739805e-01, 3.74526352e-01],
[4.57397243e-03, 4.47242975e-01, 5.48183024e-01],
[6.34019002e-02, 8.55660856e-01, 8.09372589e-02],
[8.93595040e-01, 1.06296614e-01, 1.08304048e-04],
[2.52241443e-04, 2.60304302e-01, 7.39443421e-01],
[9.14900541e-01, 8.49870592e-02, 1.12303525e-04],
[7.49440910e-03, 6.51612937e-01, 3.40892643e-01],
[9.50633347e-01, 4.93349321e-02, 3.17658523e-05],
[9.59253237e-02, 8.14125180e-01, 8.99495557e-02],
[2.11781263e-03, 2.14837521e-01, 7.83044696e-01],
[1.62444264e-02, 8.25160682e-01, 1.58594936e-01],
[8.35741055e-04, 3.18071902e-01, 6.81092441e-01],
[9.25243855e-01, 7.46982545e-02, 5.79556327e-05],
[1.80392209e-02, 5.57738781e-01, 4.24221992e-01],
[3.66105847e-02, 7.58015275e-01, 2.05374062e-01],
[9.45574045e-01, 5.43801486e-02, 4.58563263e-05],
[9.34617996e-01, 6.53414205e-02, 4.05819083e-05],
[2.17764452e-02, 7.76680112e-01, 2.01543465e-01],
[2.24324483e-02, 8.12595785e-01, 1.64971843e-01],
[6.37967363e-02, 8.47929299e-01, 8.82739723e-02],
[1.04995375e-03, 4.25696850e-01, 5.73253155e-01],
[1.56047265e-03, 1.82425112e-01, 8.16014409e-01],
[1.74497766e-03, 2.77918786e-01, 7.20336258e-01],
[5.30185699e-02, 7.86523402e-01, 1.60458028e-01],
[8.72346401e-01, 1.27430946e-01, 2.22693052e-04],
[3.79191083e-03, 5.81396639e-01, 4.14811462e-01],
[9.25170004e-01, 7.47736841e-02, 5.63827125e-05],
[6.39459398e-03, 4.11438972e-01, 5.82166433e-01],
[3.96478064e-02, 8.86551499e-01, 7.38007352e-02],
[9.79624748e-01, 2.03702692e-02, 5.04015361e-06],
[7.74072856e-03, 5.67036927e-01, 4.25222337e-01],
[2.30540000e-02, 8.39772224e-01, 1.37173772e-01],
[9.32191551e-01, 6.77113235e-02, 9.71099507e-05],
[2.72360202e-02, 7.57197618e-01, 2.15566382e-01],
[8.74242604e-01, 1.25481978e-01, 2.75358645e-04],
[1.35109825e-02, 5.62134147e-01, 4.24354821e-01],
[4.25150944e-03, 3.01454812e-01, 6.94293737e-01],
[9.29473937e-01, 7.04392493e-02, 8.69029464e-05],
[8.88650477e-01, 1.11217991e-01, 1.31544090e-04],
[3.04695275e-02, 8.62501740e-01, 1.07028767e-01],
[1.01169036e-03, 2.24886477e-01, 7.74101853e-01],
[3.04694381e-03, 4.73675370e-01, 5.23277700e-01],
[1.13608807e-01, 8.43598127e-01, 4.27931175e-02],
[9.06254709e-01, 9.36172456e-02, 1.28061511e-04],
[3.14499647e-03, 4.31593955e-01, 5.65261066e-01],
[9.34547722e-01, 6.54155836e-02, 3.66398781e-05],
[5.07650450e-02, 7.84679115e-01, 1.64555922e-01],
[1.79836333e-01, 7.73426890e-01, 4.67367582e-02],
[9.53067243e-01, 4.69031148e-02, 2.96822072e-05],
[9.60238278e-03, 7.22374380e-01, 2.68023223e-01],
[4.67043146e-02, 8.45125377e-01, 1.08170338e-01],
[3.71562541e-02, 7.69406140e-01, 1.93437621e-01],
[1.34794936e-02, 7.25503802e-01, 2.61016697e-01],
[4.23547486e-03, 4.26773846e-01, 5.68990707e-01],
[5.06713707e-03, 5.19755661e-01, 4.75177199e-01],
[1.67139655e-03, 2.41421863e-01, 7.56906748e-01],
[3.75755364e-04, 2.61969060e-01, 7.37655222e-01],
[6.80648610e-02, 7.82772779e-01, 1.49162352e-01],
[9.78763402e-01, 2.12316755e-02, 4.87002990e-06],
[1.15022771e-02, 5.55112123e-01, 4.33385640e-01],
[9.18129444e-01, 8.17450508e-02, 1.25569670e-04],
```

```
        [2.60006096e-02, 5.91036558e-01, 3.82962763e-01],
        [3.62345353e-02, 7.61694312e-01, 2.02071175e-01],
        [9.15064037e-01, 8.48172456e-02, 1.18659424e-04],
        [9.50267851e-01, 4.97112647e-02, 2.09455648e-05],
        [1.06419828e-02, 5.97191811e-01, 3.92166197e-01],
        [9.43122625e-01, 5.68432026e-02, 3.41363993e-05]], dtype=float32)
```

In [ ]:

```

```

4、图像处理工具PIL

In [ ]:

```

```