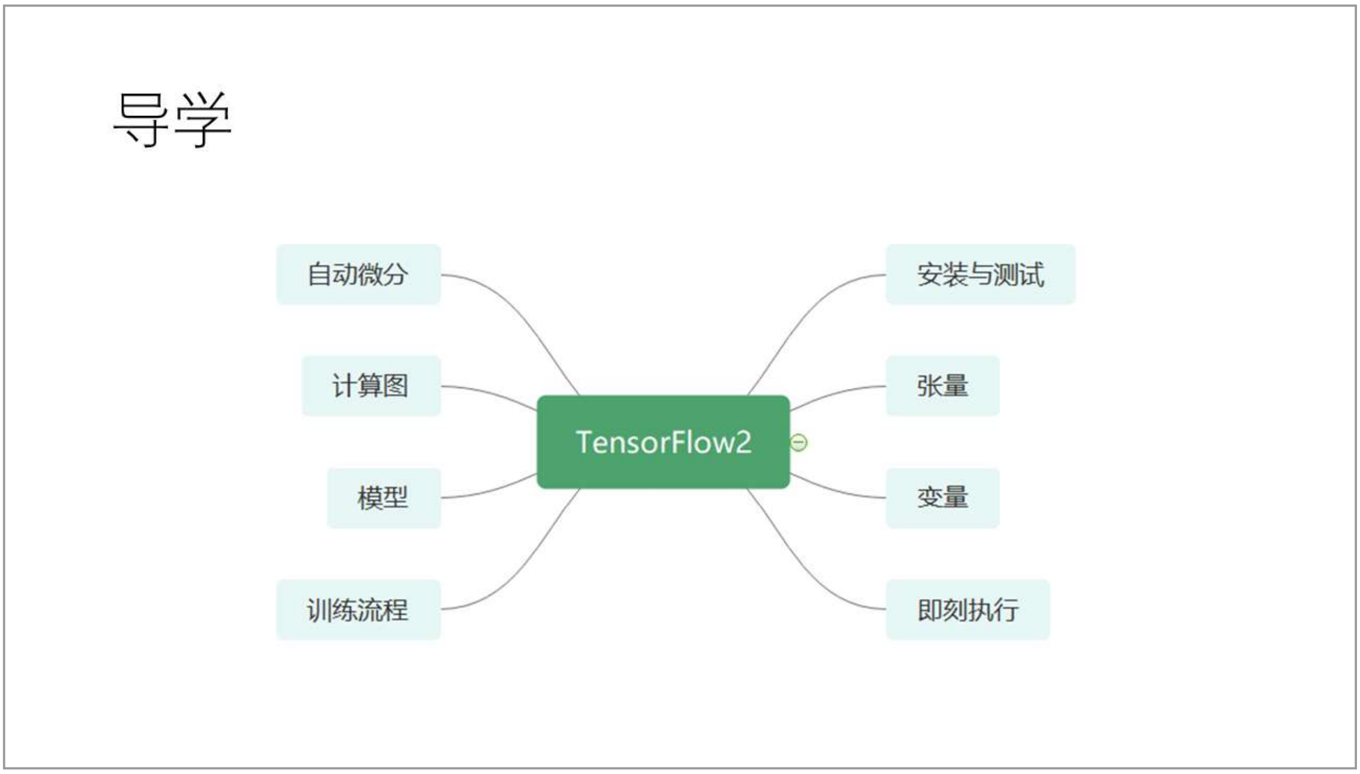
TensorFlow2

模块、层、模型

Modules\Layers\Model

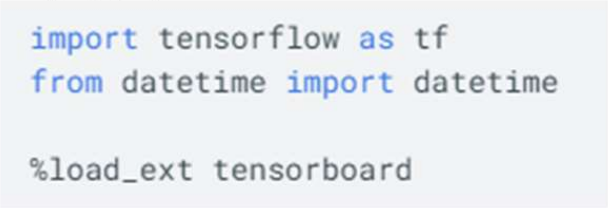


导学

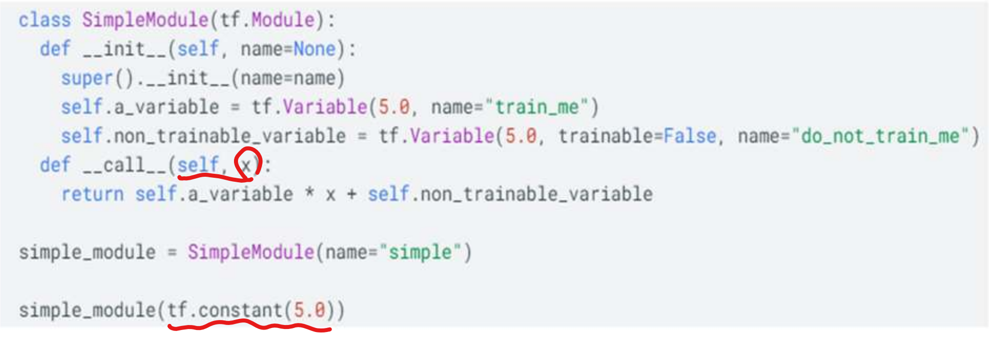
模块、层和模型

- 模块（定义、保存、恢复）
  - 对张量进行计算的函数（前向运算）
  - 变量在训练过程中被更新
- 在TensorFlow中定义模型和层
  - 层：可重用的带参数的结构
  - 所有模型和层都是 tf.Module的派生类
- tensorboard:对TensorFlow模型和张量进行可视化的工具

模块



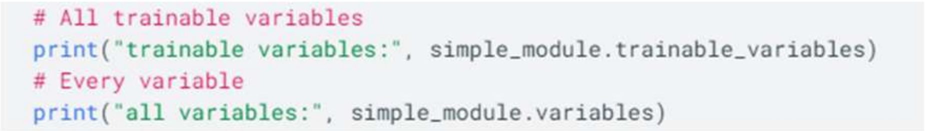
```
import tensorflow as tf
from datetime import datetime

%load_ext tensorboard
```



```
class SimpleModule(tf.Module):
  def __init__(self, name=None):
    super().__init__(name=name)
    self.a_variable = tf.Variable(5.0, name="train_me")
    self.non_trainable_variable = tf.Variable(5.0, trainable=False, name="do_not_train_me")
  def __call__(self, x):
    return self.a_variable * x + self.non_trainable_variable

simple_module = SimpleModule(name="simple")

simple_module(tf.constant(5.0))
```

a*x + b

模块



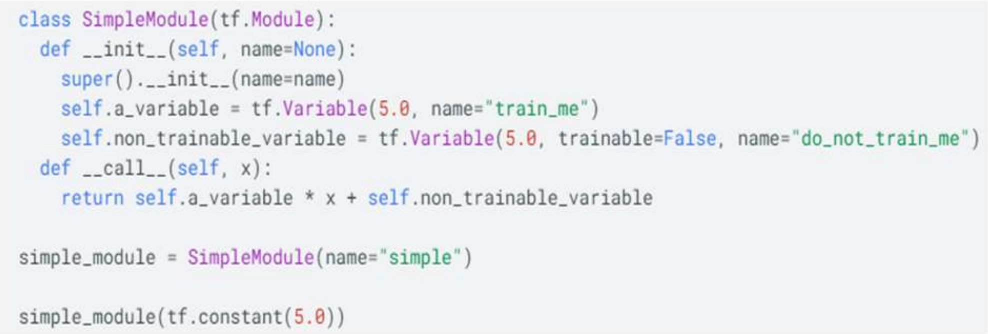
```
# All trainable variables
print("trainable variables:", simple_module.trainable_variables)
# Every variable
print("all variables:", simple_module.variables)
```

可训练变量

所有变量

```
trainable variables: (<tf.Variable 'train_me:0' shape=() dtype=float32, numpy=5.0>,)
all variables: (<tf.Variable 'train_me:0' shape=() dtype=float32, numpy=5.0>,
<tf.Variable 'do_not_train_me:0' shape=() dtype=float32, numpy=5.0>)
```

模块



```
class SimpleModule(tf.Module):
  def __init__(self, name=None):
    super().__init__(name=name)
    self.a_variable = tf.Variable(5.0, name="train_me")
    self.non_trainable_variable = tf.Variable(5.0, trainable=False, name="do_not_train_me")
  def __call__(self, x):
    return self.a_variable * x + self.non_trainable_variable

simple_module = SimpleModule(name="simple")

simple_module(tf.constant(5.0))
```

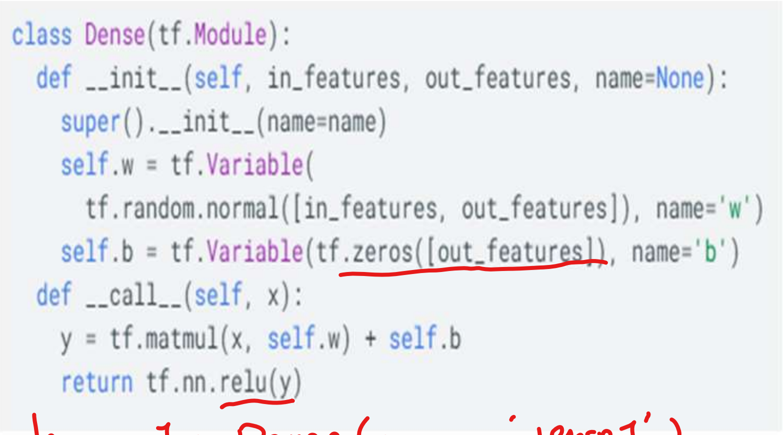
SimpleModule: 类

simple_module: 对象（实例）

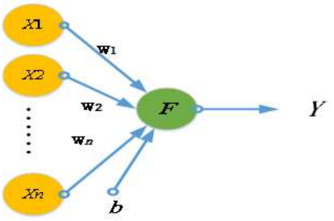
simple_module(tf.constant(5.0)): SimpleModule.__call__(simple_module, tf.constant(5.0))

相当于调用 call 函数

层



```
class Dense(tf.Module):
  def __init__(self, in_features, out_features, name=None):
    super().__init__(name=name)
    self.w = tf.Variable(
      tf.random.normal([in_features, out_features]), name='w')
    self.b = tf.Variable(tf.zeros([out_features]), name='b')
  def __call__(self, x):
    y = tf.matmul(x, self.w) + self.b
    return tf.nn.relu(y)
```



全连接层

~~W*x + b~~

xw + b

1×n n×1 1×1

dense_1 = Dense(name='dense1')

从模块到层：全连接层

dense_1(tf.constant([[2, 3, 4]]))

x 输入

想一想，练一练

请同学们投稿一下

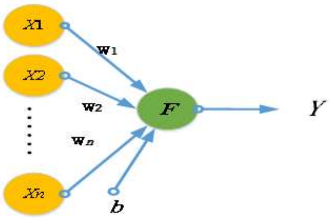
基于Dense层实现一个逻辑回归单元

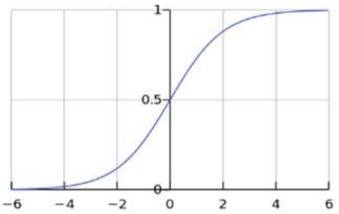
提示：激活函数为sigmoid函数

权重：[-3.14, -2.31, 2.16]

输入：[身高，体重，发长] = [0.0288, -0.3256, 0.5925]

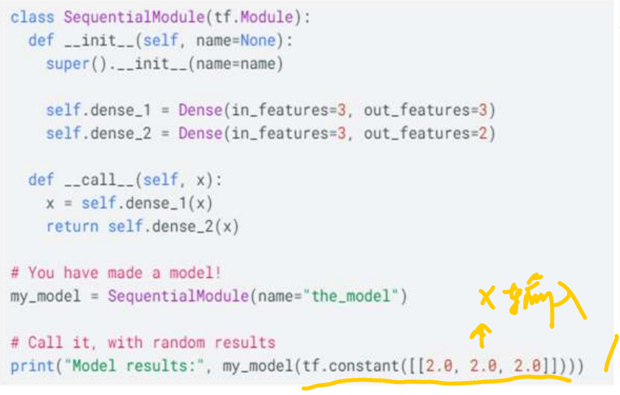
输出：猜测的概率





模型

从层到模型

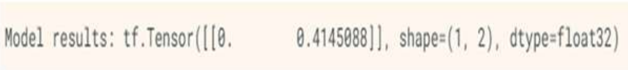


```
class SequentialModule(tf.Module):
  def __init__(self, name=None):
    super().__init__(name=name)

    self.dense_1 = Dense(in_features=3, out_features=3)
    self.dense_2 = Dense(in_features=3, out_features=2)

  def __call__(self, x):
    x = self.dense_1(x)
    return self.dense_2(x)

# You have made a model!
my_model = SequentialModule(name="the_model")

# Call it, with random results
print("Model results:", my_model(tf.constant([[2.0, 2.0, 2.0]])))
```

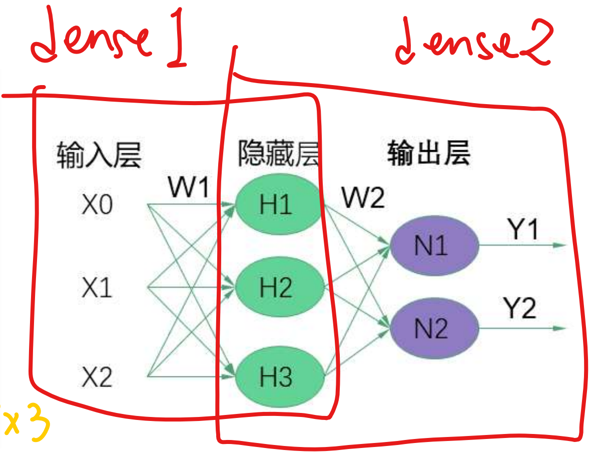
x 输入

↑

/x3

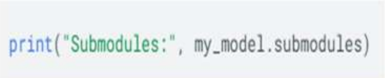


```
Model results: tf.Tensor([[0.        0.4145088]], shape=(1, 2), dtype=float32)
```

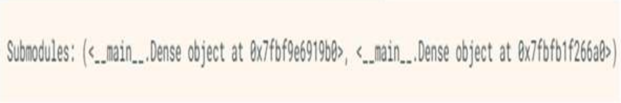


搭建两层模型模型

模型

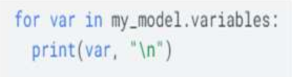


```
print("Submodules:", my_model.submodules)
```

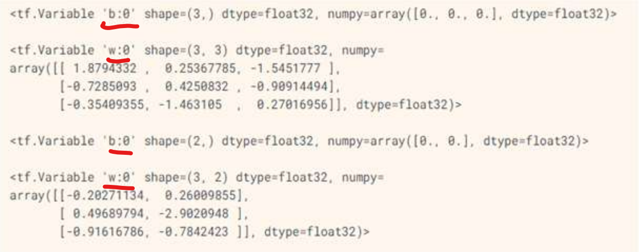


```
Submodules: (<__main__.Dense object at 0x7fbf9e6919b0>, <__main__.Dense object at 0x7fbfb1f266a0>)
```

my_model.submodules
查看模型内的子模块



```
for var in my_model.variables:
  print(var, "\n")
```



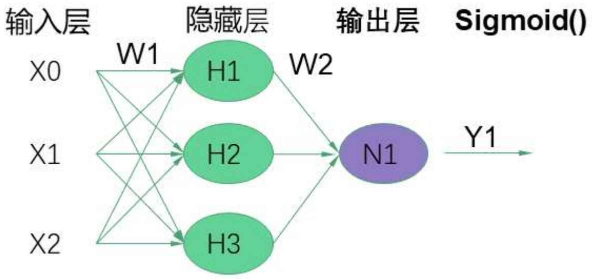
```
<tf.Variable 'b:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.], dtype=float32)>

<tf.Variable 'w:0' shape=(3, 3) dtype=float32, numpy=
array([[ 1.8794332 ,  0.25367785, -1.5451777 ],
       [-0.7285093 ,  0.4250832 , -0.90914494],
       [-0.35409355, -1.463105  ,  0.27016956]], dtype=float32)>

<tf.Variable 'b:0' shape=(2,) dtype=float32, numpy=array([0., 0.], dtype=float32)>

<tf.Variable 'w:0' shape=(3, 2) dtype=float32, numpy=
array([[-0.20271134,  0.26009855],
       [ 0.49689794, -2.9020948 ],
       [-0.91616786, -0.7842423 ]], dtype=float32)>
```

my_model.variables
查看模型内的变量

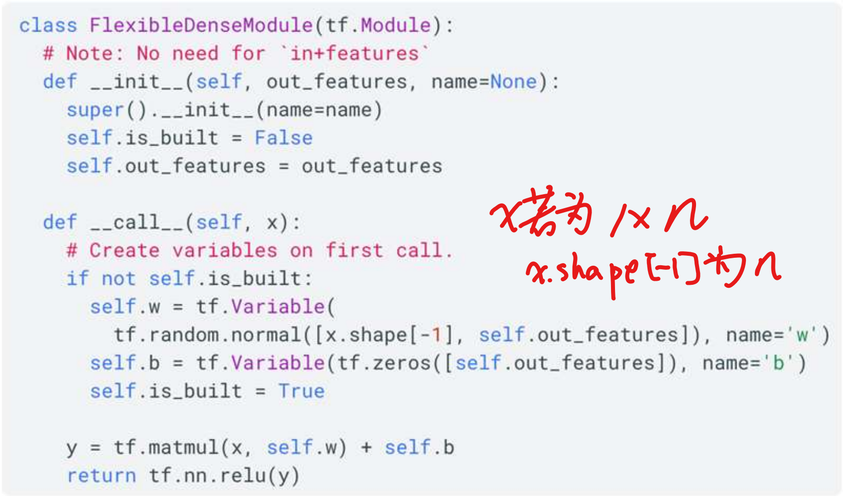
想一想，练一练

- 请同学们投稿一下
- 实现两层全连接预测男生/女生

- 输入：[身高，体重，发长] = [0.0288, -0.3256, 0.5925]
- 输出：猜测的概率



动态决定张量维度（补充）

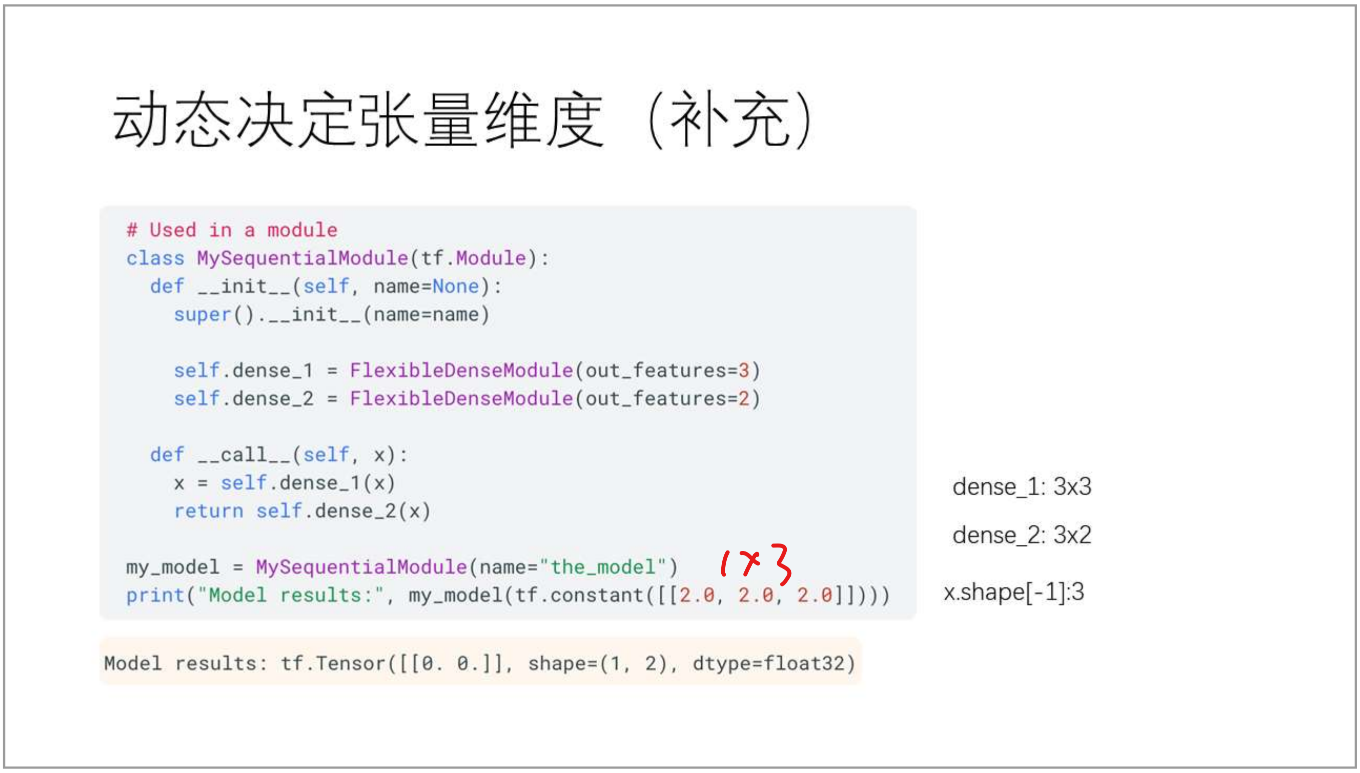


```
class FlexibleDenseModule(tf.Module):
  # Note: No need for `in+features`
  def __init__(self, out_features, name=None):
    super().__init__(name=name)
    self.is_built = False
    self.out_features = out_features

  def __call__(self, x):
    # Create variables on first call.
    if not self.is_built:
      self.w = tf.Variable(
        tf.random.normal([x.shape[-1], self.out_features]), name='w')
      self.b = tf.Variable(tf.zeros([self.out_features]), name='b')
      self.is_built = True

    y = tf.matmul(x, self.w) + self.b
    return tf.nn.relu(y)
```

x若为 1×n
x.shape[-1]为n

is_built

x.shape[-1]



动态决定张量维度（补充）

```python
# Used in a module
class MySequentialModule(tf.Module):
  def __init__(self, name=None):
    super().__init__(name=name)

    self.dense_1 = FlexibleDenseModule(out_features=3)
    self.dense_2 = FlexibleDenseModule(out_features=2)

  def __call__(self, x):
    x = self.dense_1(x)
    return self.dense_2(x)

my_model = MySequentialModule(name="the_model")
print("Model results:", my_model(tf.constant([[2.0, 2.0, 2.0]])))
```

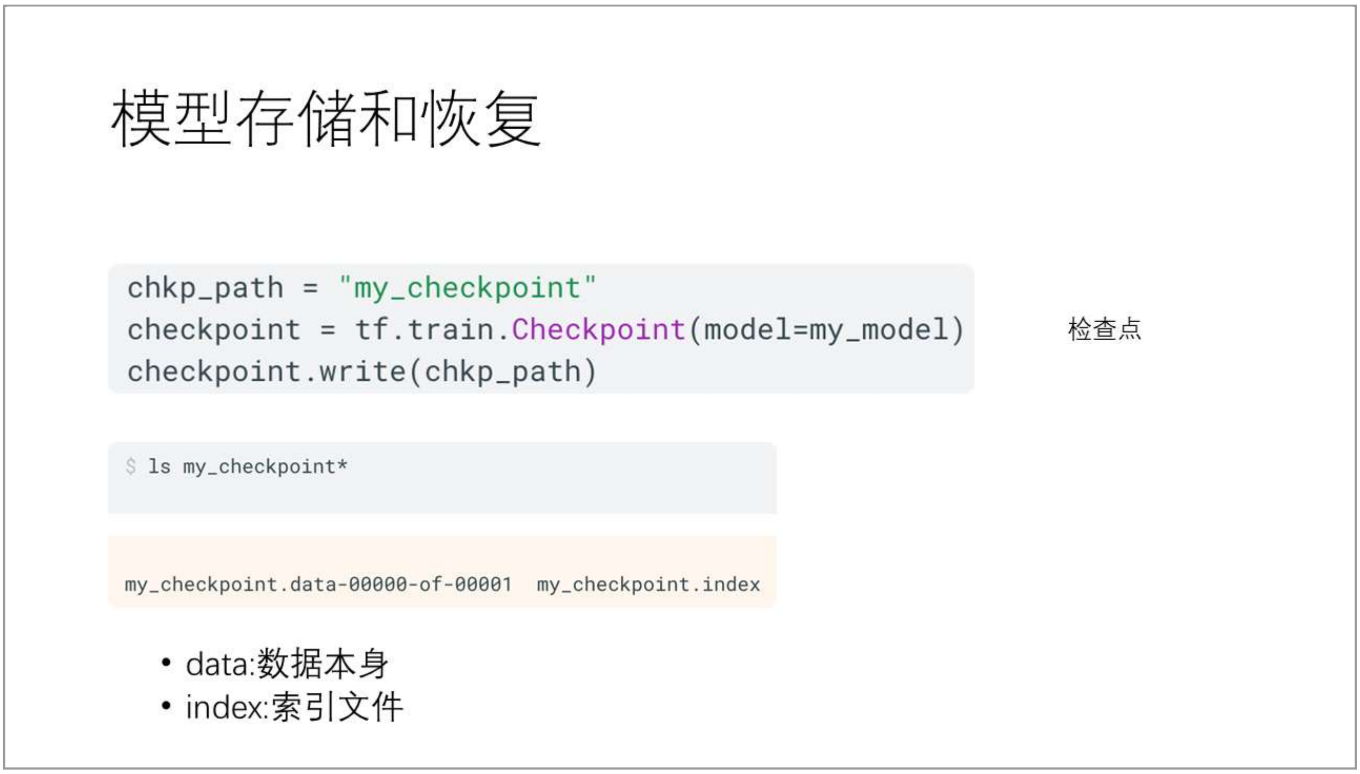
1×3

```
Model results: tf.Tensor([[0. 0.]], shape=(1, 2), dtype=float32)
```

dense_1: 3x3

dense_2: 3x2

x.shape[-1]:3



模型存储和恢复

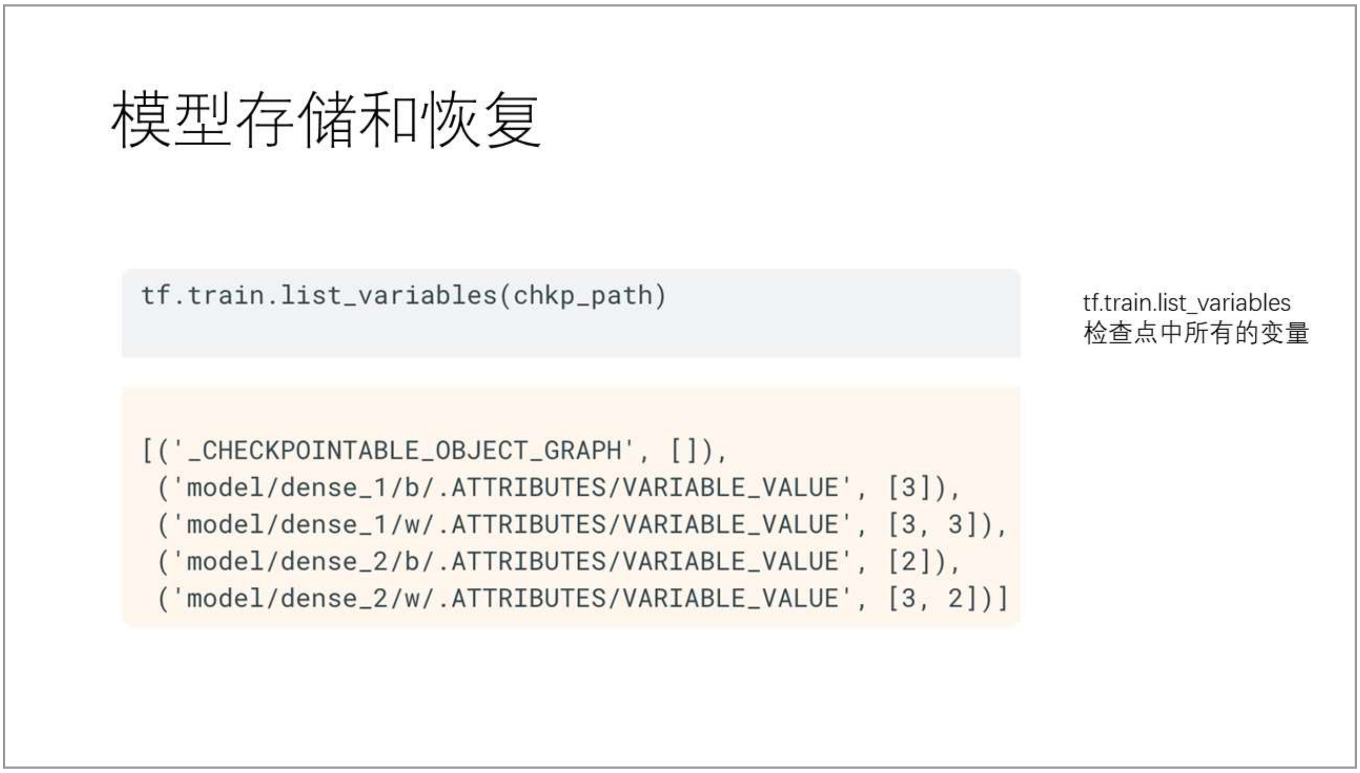
```python
chkp_path = "my_checkpoint"
checkpoint = tf.train.Checkpoint(model=my_model)
checkpoint.write(chkp_path)
```

检查点

```
$ ls my_checkpoint*
```

```
my_checkpoint.data-00000-of-00001  my_checkpoint.index
```

- data:数据本身
- index:索引文件

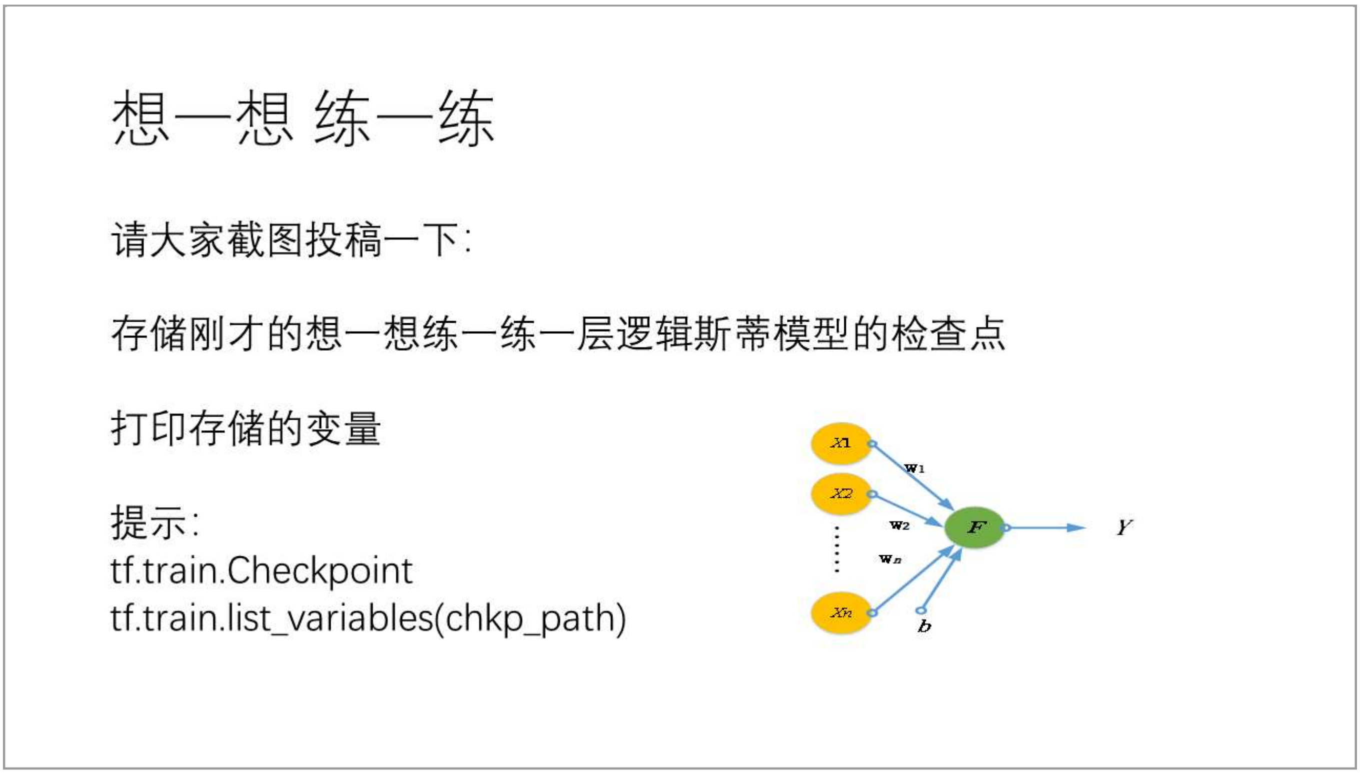


模型存储和恢复

```
tf.train.list_variables(chkp_path)
```

```
[('_CHECKPOINTABLE_OBJECT_GRAPH', []),
 ('model/dense_1/b/.ATTRIBUTES/VARIABLE_VALUE', [3]),
 ('model/dense_1/w/.ATTRIBUTES/VARIABLE_VALUE', [3, 3]),
 ('model/dense_2/b/.ATTRIBUTES/VARIABLE_VALUE', [2]),
 ('model/dense_2/w/.ATTRIBUTES/VARIABLE_VALUE', [3, 2])]
```

tf.train.list_variables
检查点中所有的变量



想一想 练一练

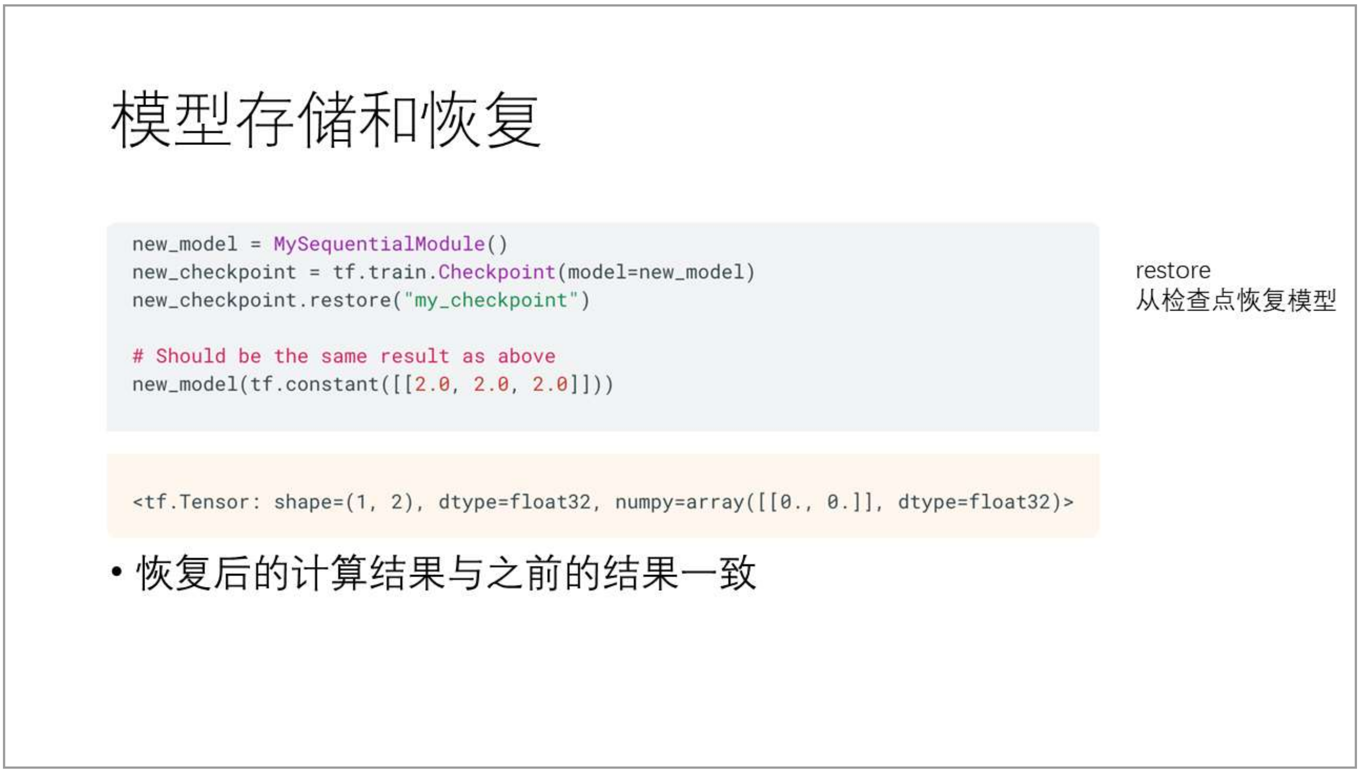
请大家截图投稿一下:

存储刚才的想一想练一练一层逻辑斯蒂模型的检查点

打印存储的变量

提示:
tf.train.Checkpoint
tf.train.list_variables(chkp_path)

X1
X2
Xn
w1
w2
wn
F
Y
b



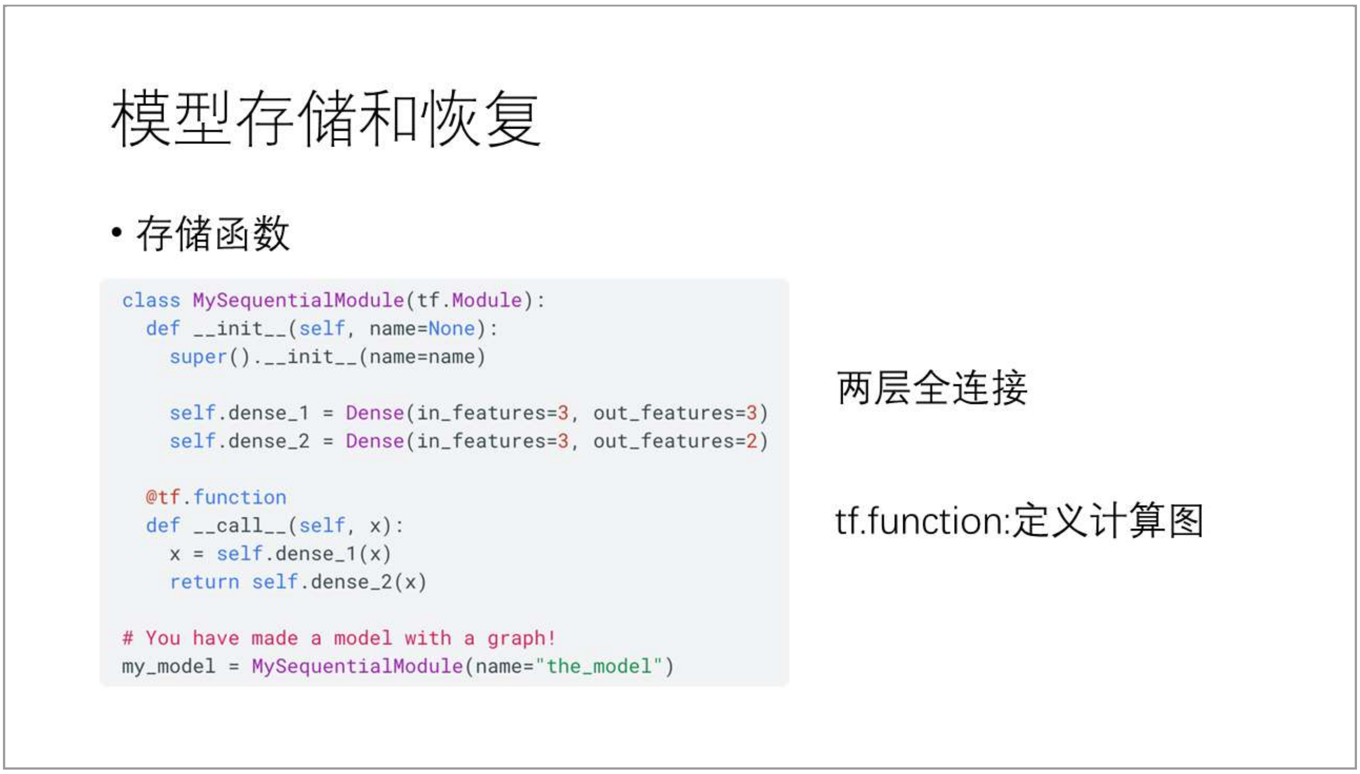
模型存储和恢复

```
new_model = MySequentialModule()
new_checkpoint = tf.train.Checkpoint(model=new_model)
new_checkpoint.restore("my_checkpoint")

# Should be the same result as above
new_model(tf.constant([[2.0, 2.0, 2.0]]))
```

```
<tf.Tensor: shape=(1, 2), dtype=float32, numpy=array([[0., 0.]], dtype=float32)>
```

restore
从检查点恢复模型

- 恢复后的计算结果与之前的结果一致



模型存储和恢复

- 存储函数

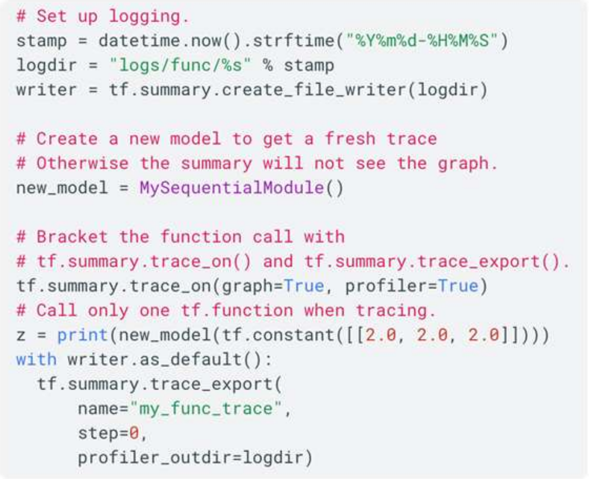
```
class MySequentialModule(tf.Module):
  def __init__(self, name=None):
    super().__init__(name=name)

    self.dense_1 = Dense(in_features=3, out_features=3)
    self.dense_2 = Dense(in_features=3, out_features=2)

  @tf.function
  def __call__(self, x):
    x = self.dense_1(x)
    return self.dense_2(x)

# You have made a model with a graph!
my_model = MySequentialModule(name="the_model")
```

两层全连接

tf.function:定义计算图

模型存储和恢复

- tensorboard可视化



```
# Set up logging.
stamp = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = "logs/func/%s" % stamp
writer = tf.summary.create_file_writer(logdir)

# Create a new model to get a fresh trace
# Otherwise the summary will not see the graph.
new_model = MySequentialModule()

# Bracket the function call with
# tf.summary.trace_on() and tf.summary.trace_export().
tf.summary.trace_on(graph=True, profiler=True)
# Call only one tf.function when tracing.
z = print(new_model(tf.constant([[2.0, 2.0, 2.0]])))
with writer.as_default():
  tf.summary.trace_export(
      name="my_func_trace",
      step=0,
      profiler_outdir=logdir)
```

create_file_writer指定输出的路径

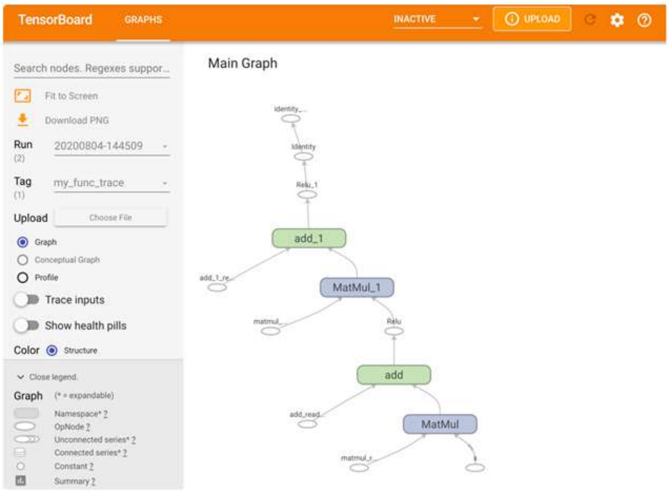
trace_on开始记录计算图

trace_export停止记录并把之前的记录导出

模型存储和恢复

- tensorboard可视化

在jupyter notebook中：%tensorboard --logdir logs/func



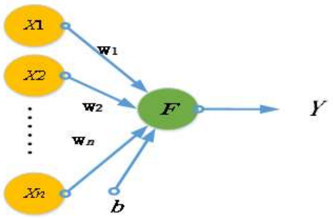
Run (2) 20200804-144509

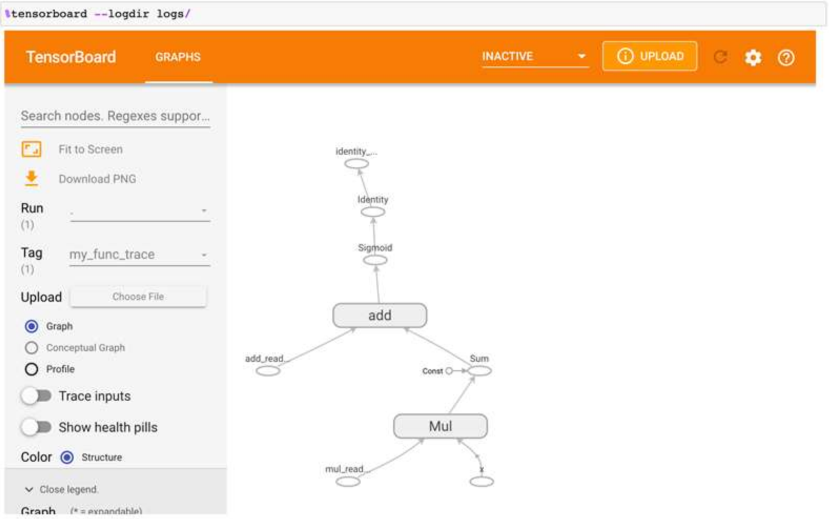
Tag (1) my_func_trace

想一想， 练一练

使用tensorboard可视化一层逻辑斯蒂回归单元

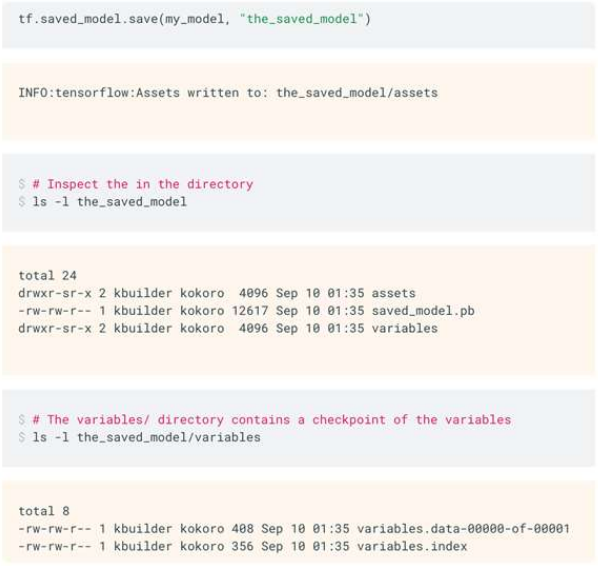
请同学们投稿一下





```
tensorboard --logdir logs/
```

存储整个模型



```
tf.saved_model.save(my_model, "the_saved_model")
```

模型导出

```
INFO:tensorflow:Assets written to: the_saved_model/assets
```

```
$ # Inspect the in the directory
$ ls -l the_saved_model
```

```
total 24
drwxr-sr-x 2 kbuilder kokoro  4096 Sep 10 01:35 assets
-rw-rw-r-- 1 kbuilder kokoro 12617 Sep 10 01:35 saved_model.pb
drwxr-sr-x 2 kbuilder kokoro  4096 Sep 10 01:35 variables
```

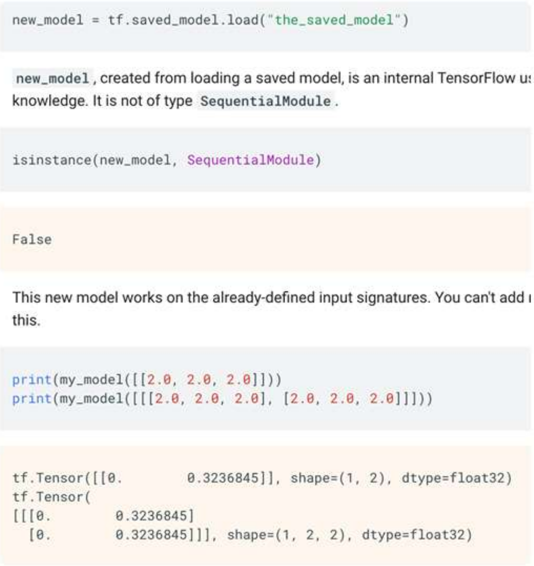
saved_model.pb文件描述了tf.Graph

```
$ # The variables/ directory contains a checkpoint of the variables
$ ls -l the_saved_model/variables
```

```
total 8
-rw-rw-r-- 1 kbuilder kokoro 408 Sep 10 01:35 variables.data-00000-of-00001
-rw-rw-r-- 1 kbuilder kokoro 356 Sep 10 01:35 variables.index
```

数据和索引文件

恢复整个模型



```
new_model = tf.saved_model.load("the_saved_model")
```

导入存储的模型

`new_model`, created from loading a saved model, is an internal TensorFlow us
knowledge. It is not of type `SequentialModule`.

```
isinstance(new_model, SequentialModule)
```

```
False
```

恢复的模型不是原来的SequentialModule类

This new model works on the already-defined input signatures. You can't add
this.

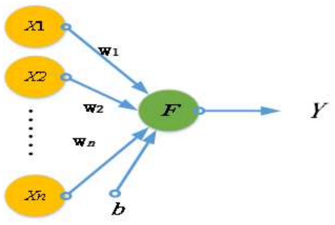
```
print(my_model([[2.0, 2.0, 2.0]]))
print(my_model([[[2.0, 2.0, 2.0], [2.0, 2.0, 2.0]]]))
```

```
tf.Tensor([[0.        0.3236845]], shape=(1, 2), dtype=float32)
tf.Tensor(
[[[0.        0.3236845]
  [0.        0.3236845]]], shape=(1, 2, 2), dtype=float32)
```

可以直接使用进行前向运算

想一想， 练一练

将一层逻辑斯蒂回归模型导出，路径名为bdmi_model,
再重新导入
用输入为 [3.0,3.0,3.0]
请同学们将前向计算的结果投稿一下



谢谢指正！