TensorFlow2教程-Keras概述

Keras 是一個用於構建和訓練深度學習模型的高階 API。它可用於快速設計原型、高級研究和生產。

Keras的3個優點： 方便使用者使用、模組化和可組合、易於擴展

1 導入tf.keras

TensorFlow2推薦使用tf.keras構建網路，常見的神經網路都包含在tf.keras.layer中(最新的tf.keras的版本可能和keras不同)

In [1]:
```
import tensorflow as tf
from tensorflow.keras import layers
print(tf.__version__)
print(tf.keras.__version__)
```

```
2.3.1
2.4.0
```

2 構建簡單模型

2.1 模型堆疊疊

常見的模型類型是層的堆疊：tf.keras.Sequential 模型

In [2]:
```
model = tf.keras.Sequential()
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

2.2 網路配置

tf.keras.layers中主要的網路配置參數如下：

activation：設置層的啟動函數。此參數可以是函數名稱字串，也可以是函數物件。預設情況下，系統不會應用任何啟動函數。

kernel_initializer 和 bias_initializer：創建層權重（核和偏置）的初始化方案。此參數是一個名稱或可調用的函數物件，預設為 "Glorot uniform" 初始化器。

kernel_regularizer 和 bias_regularizer：應用層權重（核和偏置）的正則化方案，例如 L1 或 L2 正則化。預設情況下，系統不會應用正則化函數。

In [3]:
```
layers.Dense(32, activation='sigmoid')
layers.Dense(32, activation=tf.sigmoid)
layers.Dense(32, kernel_initializer='orthogonal')
layers.Dense(32, kernel_initializer=tf.keras.initializers.glorot_normal)
layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01))
layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l1(0.01))
```

Out[3]:
```
<tensorflow.python.keras.layers.core.Dense at 0x1a912c7ee48>
```

3 訓練和評估

3.1 設置訓練流程

構建好模型後，通過調用 compile 方法配置該模型的學習流程

In [4]:
```
model = tf.keras.Sequential()
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=[tf.keras.metrics.categorical_accuracy])
```

3.2 輸入Numpy資料

對於小型資料集，可以使用Numpy構建輸入資料。

In [5]:
```
import numpy as np

train_x = np.random.random((1000, 72))
train_y = np.random.random((1000, 10))

val_x = np.random.random((200, 72))
val_y = np.random.random((200, 10))

model.fit(train_x, train_y, epochs=10, batch_size=100,
          validation_data=(val_x, val_y))
```

```
Epoch 1/10
10/10 [==============================] - 0s 14ms/step - loss: 11.7214 - categ
orical_accuracy: 0.0900 - val_loss: 11.8297 - val_categorical_accuracy: 0.115
0
Epoch 2/10
10/10 [==============================] - 0s 3ms/step - loss: 11.9965 - catego
rical_accuracy: 0.1020 - val_loss: 12.4094 - val_categorical_accuracy: 0.1250
Epoch 3/10
10/10 [==============================] - 0s 3ms/step - loss: 12.9817 - catego
rical_accuracy: 0.0900 - val_loss: 14.0708 - val_categorical_accuracy: 0.1350
Epoch 4/10
10/10 [==============================] - 0s 3ms/step - loss: 15.3001 - catego
rical_accuracy: 0.0930 - val_loss: 17.1503 - val_categorical_accuracy: 0.1300
Epoch 5/10
10/10 [==============================] - 0s 3ms/step - loss: 18.8359 - catego
rical_accuracy: 0.0930 - val_loss: 21.1884 - val_categorical_accuracy: 0.1300
Epoch 6/10
10/10 [==============================] - 0s 3ms/step - loss: 23.2201 - catego
rical_accuracy: 0.0930 - val_loss: 25.9848 - val_categorical_accuracy: 0.1250
```

3.3 tf.data輸入資料

對於大型資料集可以使用tf.data構建訓練輸入。

In [6]:
```
dataset = tf.data.Dataset.from_tensor_slices((train_x, train_y))
dataset = dataset.batch(32)
dataset = dataset.repeat()
val_dataset = tf.data.Dataset.from_tensor_slices((val_x, val_y))
val_dataset = val_dataset.batch(32)
val_dataset = val_dataset.repeat()

model.fit(dataset, epochs=10, steps_per_epoch=30,
          validation_data=val_dataset, validation_steps=3)
```

```
Epoch 1/10
WARNING:tensorflow:Layer dense_9 is casting an input tensor from dtype float64
to the layer's dtype of float32, which is new behavior in TensorFlow 2.  The la
yer has dtype float32 because its dtype defaults to floatx.

If you intended to run this layer in float32, you can safely ignore this warnin
g. If in doubt, this warning is likely only an issue if you are porting a Tenso
rFlow 1.X model to TensorFlow 2.

To change all layers to have dtype float64 by default, call `tf.keras.backend.s
et_floatx('float64')`. To change just this layer, pass dtype='float64' to the l
ayer constructor. If you are the author of this layer, you can disable autocast
ing by passing autocast=False to the base Layer constructor.

30/30 [==============================] - 0s 6ms/step - loss: 83.1468 - categori
cal_accuracy: 0.0896 - val_loss: 106.3833 - val_categorical_accuracy: 0.1458
Epoch 2/10
30/30 [==============================] - 0s 2ms/step - loss: 139.7436 - categor
ical_accuracy: 0.0972 - val_loss: 170.3425 - val_categorical_accuracy: 0.1458
Epoch 3/10
30/30 [==============================] - 0s 2ms/step - loss: 213.0894 - categor
ical_accuracy: 0.0908 - val_loss: 251.1199 - val_categorical_accuracy: 0.1562
Epoch 4/10
30/30 [==============================] - 0s 2ms/step - loss: 302.7543 - categor
ical_accuracy: 0.0983 - val_loss: 345.4188 - val_categorical_accuracy: 0.1146
Epoch 5/10
30/30 [==============================] - 0s 2ms/step - loss: 400.2596 - categor
ical_accuracy: 0.0919 - val_loss: 443.5746 - val_categorical_accuracy: 0.1250
Epoch 6/10
30/30 [==============================] - 0s 2ms/step - loss: 503.8910 - categor
ical_accuracy: 0.0962 - val_loss: 546.4531 - val_categorical_accuracy: 0.1458
Epoch 7/10
30/30 [==============================] - 0s 2ms/step - loss: 609.4665 - categor
ical_accuracy: 0.1015 - val_loss: 649.1159 - val_categorical_accuracy: 0.1146
Epoch 8/10
30/30 [==============================] - 0s 2ms/step - loss: 710.9148 - categor
ical_accuracy: 0.1111 - val_loss: 737.7017 - val_categorical_accuracy: 0.1458
Epoch 9/10
30/30 [==============================] - 0s 2ms/step - loss: 780.7780 - categor
ical_accuracy: 0.1165 - val_loss: 782.8152 - val_categorical_accuracy: 0.0521
Epoch 10/10
30/30 [==============================] - 0s 2ms/step - loss: 797.5306 - categor
ical_accuracy: 0.1122 - val_loss: 752.1121 - val_categorical_accuracy: 0.1250
```

Out[6]:
```
<tensorflow.python.keras.callbacks.History at 0x1a914019388>
```

3.4 評估與預測

評估和預測函數：tf.keras.Model.evaluate和tf.keras.Model.predict方法，都可以可以使用NumPy和tf.data.Dataset構造的輸入資料進行評估和預測

In [7]:
```
# 模型评估
test_x = np.random.random((1000, 72))
test_y = np.random.random((1000, 10))
model.evaluate(test_x, test_y, batch_size=32)
test_data = tf.data.Dataset.from_tensor_slices((test_x, test_y))
test_data = test_data.batch(32).repeat()
model.evaluate(test_data, steps=30)
```

```
32/32 [==============================] - 0s 1ms/step - loss: 776.9584 - categor
ical_accuracy: 0.1000
30/30 [==============================] - 0s 1ms/step - loss: 776.5385 - categor
ical_accuracy: 0.1000
```

Out[7]:
```
[776.5384521484375, 0.10000000149011612]
```

In [8]:
```
# 模型預測
result = model.predict(test_x, batch_size=32)
print(result)
```

```
[[2.7464208e-01 0.0000000e+00 0.0000000e+00 ... 1.9201213e-01
  0.0000000e+00 0.0000000e+00]
 [4.0363491e-01 0.0000000e+00 0.0000000e+00 ... 1.3699500e-01
  0.0000000e+00 0.0000000e+00]
 [3.4372190e-01 0.0000000e+00 0.0000000e+00 ... 1.3705732e-01
  0.0000000e+00 0.0000000e+00]
 ...
 [5.4296976e-01 0.0000000e+00 0.0000000e+00 ... 1.1221240e-01
  0.0000000e+00 0.0000000e+00]
 [3.2302347e-01 0.0000000e+00 3.3144156e-38 ... 1.5574698e-01
  0.0000000e+00 0.0000000e+00]
 [3.5760310e-01 0.0000000e+00 1.6393766e-38 ... 1.3908535e-01
  0.0000000e+00 0.0000000e+00]]
```

4 構建複雜模型

4.1 函数式API

tf.keras.Sequential 模型是層的簡單堆疊，無法表示任意模型。使用 Keras的函數式API可以構建複雜的模型拓撲，例如：

- 多輸入模型，
- 多輸出模型，
- 具有共用層的模型（同一層被調用多次），
- 具有非序列資料流程的模型（例如，殘差連接）。

使用函數式 API 構建的模型具有以下特徵：

- 層實例可調用並返回張量。

- 輸入張量和輸出張量用於定義 tf.keras.Model 實例。
- 此模型的訓練方式和 Sequential 模型一樣。

In [9]:
```
input_x = tf.keras.Input(shape=(72,))
hidden1 = layers.Dense(32, activation='relu')(input_x)
hidden2 = layers.Dense(16, activation='relu')(hidden1)
pred = layers.Dense(10, activation='softmax')(hidden2)
# 構建tf.keras.Model實例
model = tf.keras.Model(inputs=input_x, outputs=pred)
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=['accuracy'])
model.fit(train_x, train_y, batch_size=32, epochs=5)
```

```
Epoch 1/5
32/32 [==============================] - 0s 1ms/step - loss: 14.0960 - accurac
y: 0.1240
Epoch 2/5
32/32 [==============================] - 0s 1ms/step - loss: 25.5531 - accurac
y: 0.1090
Epoch 3/5
32/32 [==============================] - 0s 1ms/step - loss: 46.0708 - accurac
y: 0.1140
Epoch 4/5
32/32 [==============================] - 0s 1ms/step - loss: 80.9490 - accurac
y: 0.1160
Epoch 5/5
32/32 [==============================] - 0s 997us/step - loss: 131.3238 - accur
acy: 0.1070
```

Out[9]:
```
<tensorflow.python.keras.callbacks.History at 0x1a915324188>
```

4.2 模型子類化

可以通過對 tf.keras.Model 進行子類化並定義自己的前向傳播來構建完全可自訂的模型。

- 在_*init*_ 方法中創建層並將它們設置為類實例的屬性。
- 在__call__方法中定義前向傳播

In [10]:
```
class MyModel(tf.keras.Model):
    def __init__(self, num_classes=10):
        super(MyModel, self).__init__(name='my_model')
        self.num_classes = num_classes
        # 定義網路層
        self.layer1 = layers.Dense(32, activation='relu')
        self.layer2 = layers.Dense(num_classes, activation='softmax')
    def call(self, inputs):
        # 定義前向傳播
        h1 = self.layer1(inputs)
        out = self.layer2(h1)
        return out

    def compute_output_shape(self, input_shape):
        # 計算輸出shape
        shape = tf.TensorShape(input_shape).as_list()
        shape[-1] = self.num_classes
        return tf.TensorShape(shape)
# 產生實體模型類，並訓練
model = MyModel(num_classes=10)
model.compile(optimizer=tf.keras.optimizers.RMSprop(0.001),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=['accuracy'])

model.fit(train_x, train_y, batch_size=16, epochs=5)
```

```
Epoch 1/5
63/63 [==============================] - 0s 775us/step - loss: 13.9589 - accura
cy: 0.1020
Epoch 2/5
63/63 [==============================] - 0s 744us/step - loss: 18.5865 - accura
cy: 0.1110
Epoch 3/5
63/63 [==============================] - 0s 841us/step - loss: 23.5508 - accura
cy: 0.1060
Epoch 4/5
63/63 [==============================] - 0s 867us/step - loss: 29.0338 - accura
cy: 0.1170
Epoch 5/5
63/63 [==============================] - 0s 1ms/step - loss: 36.0880 - accurac
y: 0.1010
```

Out[10]:
```
<tensorflow.python.keras.callbacks.History at 0x1a9163c9208>
```

4.3 自訂層

通過對 tf.keras.layers.Layer 進行子類化並實現以下方法來創建自訂層：

- *_init_*: (可選)定義該層要使用的子層
- build：創建層的權重。使用 add_weight 方法添加權重。
- call：定義前向傳播。
- compute_output_shape：指定在給定輸入形狀的情況下如何計算層的輸出形狀。

- 可選，可以通過實現 get_config 方法和 from_config 類方法序列化層。

In [11]:
```
class MyLayer(layers.Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(MyLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        shape = tf.TensorShape((input_shape[1], self.output_dim))
        self.kernel = self.add_weight(name='kernel1', shape=shape,
                                      initializer='uniform', trainable=True)
        super(MyLayer, self).build(input_shape)

    def call(self, inputs):
        return tf.matmul(inputs, self.kernel)

    def compute_output_shape(self, input_shape):
        shape = tf.TensorShape(input_shape).as_list()
        shape[-1] = self.output_dim
        return tf.TensorShape(shape)

    def get_config(self):
        base_config = super(MyLayer, self).get_config()
        base_config['output_dim'] = self.output_dim
        return base_config

    @classmethod
    def from_config(cls, config):
        return cls(**config)

# 使用自訂網路層構建模型
model = tf.keras.Sequential(
[
    MyLayer(10),
    layers.Activation('softmax')
])


model.compile(optimizer=tf.keras.optimizers.RMSprop(0.001),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=['accuracy'])

model.fit(train_x, train_y, batch_size=16, epochs=5)
```

```
Epoch 1/5
63/63 [==============================] - 0s 989us/step - loss: 11.5528 - accu
racy: 0.0930
Epoch 2/5
63/63 [==============================] - 0s 823us/step - loss: 11.5502 - accu
racy: 0.0930
Epoch 3/5
63/63 [==============================] - 0s 775us/step - loss: 11.5496 - accu
racy: 0.0950
Epoch 4/5
63/63 [==============================] - 0s 807us/step - loss: 11.5471 - accu
racy: 0.0990
```

```
Epoch 5/5
63/63 [==============================] - 0s 789us/step - loss: 11.5460 - accu
racy: 0.0930
```

Out[11]: <tensorflow.python.keras.callbacks.History at 0x1a9164efac8>

4.3 回檔

回檔是傳遞給模型以自訂和擴展其在訓練期間的行為的對象。我們可以編寫自己的自訂回檔，或使用tf.keras.callbacks中的內置函數，常用內置回呼函數如下：

- tf.keras.callbacks.ModelCheckpoint：定期保存模型的檢查點。
- tf.keras.callbacks.LearningRateScheduler：動態更改學習率。
- tf.keras.callbacks.EarlyStopping：驗證性能停止提高時進行中斷培訓。
- tf.keras.callbacks.TensorBoard：使用TensorBoard監視模型的行為 。

In [14]:
```
callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_loss'),
    tf.keras.callbacks.TensorBoard(log_dir='.\logs')
]
model.fit(train_x, train_y, batch_size=16, epochs=5,
          callbacks=callbacks, validation_data=(val_x, val_y))
```

```
Epoch 1/5
 2/63 [..............................] - ETA: 1s - loss: 12.2965 - accuracy: 0.
0625WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared t
o the batch time (batch time: 0.0020s vs `on_train_batch_end` time: 0.0568s). C
heck your callbacks.
63/63 [==============================] - 0s 3ms/step - loss: 11.5327 - accurac
y: 0.0960 - val_loss: 11.5297 - val_accuracy: 0.1300
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 11.5323 - accurac
y: 0.1030 - val_loss: 11.5276 - val_accuracy: 0.1200
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 11.5321 - accurac
y: 0.0980 - val_loss: 11.5282 - val_accuracy: 0.1250
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 11.5315 - accurac
y: 0.0920 - val_loss: 11.5268 - val_accuracy: 0.1200
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 11.5310 - accurac
y: 0.1050 - val_loss: 11.5269 - val_accuracy: 0.1200
```

Out[14]: <tensorflow.python.keras.callbacks.History at 0x1a91767b1c8>

5 模型保存與恢復

5.1 權重保存

In [15]:
```
model = tf.keras.Sequential([
layers.Dense(64, activation='relu', input_shape=(32,)),  # 需要有input_shape
layers.Dense(10, activation='softmax')])

model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [16]:
```
# 權重保存與重載
model.save_weights('.weights\My_Model')
model.load_weights('.weights\My_Model')
# 保存為h5格式
model.save_weights('.\model.h5', save_format='h5')
model.load_weights('.\model.h5')
```

5.2 保存網路結構

In [17]:
```
# 序列化成json
import json
import pprint
json_str = model.to_json()
pprint.pprint(json.loads(json_str))
# 從json中重建模型
fresh_model = tf.keras.models.model_from_json(json_str)
```

```
{'backend': 'tensorflow',
 'class_name': 'Sequential',
 'config': {'layers': [{'class_name': 'InputLayer',
                        'config': {'batch_input_shape': [None, 32],
                                   'dtype': 'float32',
                                   'name': 'dense_17_input',
                                   'ragged': False,
                                   'sparse': False}},
                       {'class_name': 'Dense',
                        'config': {'activation': 'relu',
                                   'activity_regularizer': None,
                                   'batch_input_shape': [None, 32],
                                   'bias_constraint': None,
                                   'bias_initializer': {'class_name': 'Zeros',
                                                        'config': {}},
                                   'bias_regularizer': None,
                                   'dtype': 'float32',
                                   'kernel_constraint': None,
                                   'kernel_initializer': {'class_name': 'Glorot
Uniform',
                                                          'config': {'seed': No
ne}},
                                   'kernel_regularizer': None,
                                   'name': 'dense_17',
                                   'trainable': True,
                                   'units': 64,
                                   'use_bias': True}},
                       {'class_name': 'Dense',
                        'config': {'activation': 'softmax',
                                   'activity_regularizer': None,
                                   'bias_constraint': None,
                                   'bias_initializer': {'class_name': 'Zeros',
                                                        'config': {}},
                                   'bias_regularizer': None,
                                   'dtype': 'float32',
                                   'kernel_constraint': None,
                                   'kernel_initializer': {'class_name': 'Glorot
Uniform',
                                                          'config': {'seed': No
ne}},
                                   'kernel_regularizer': None,
                                   'name': 'dense_18',
                                   'trainable': True,
                                   'units': 10,
                                   'use_bias': True}}],
            'name': 'sequential_3'},
 'keras_version': '2.4.0'}
```

In [18]:
```
# 保持為yaml格式  #需要提前安裝pyyaml

yaml_str = model.to_yaml()
print(yaml_str)
# 從yaml資料中重新構建模型
fresh_model = tf.keras.models.model_from_yaml(yaml_str)
```

```
backend: tensorflow
class_name: Sequential
config:
  layers:
  - class_name: Dense
    config:
      activation: relu
      activity_regularizer: null
      batch_input_shape: !!python/tuple
      - null
      - 32
      bias_constraint: null
      bias_initializer:
        class_name: Zeros
        config: {}
      bias_regularizer: null
      dtype: float32
      kernel_constraint: null
      kernel_initializer:
        class_name: GlorotUniform
        config:
          seed: null
      kernel_regularizer: null
      name: dense_18
      trainable: true
      units: 64
      use_bias: true
  - class_name: Dense
    config:
      activation: softmax
      activity_regularizer: null
      bias_constraint: null
      bias_initializer:
        class_name: Zeros
        config: {}
      bias_regularizer: null
      dtype: float32
      kernel_constraint: null
      kernel_initializer:
        class_name: GlorotUniform
        config:
          seed: null
      kernel_regularizer: null
      name: dense_19
      trainable: true
      units: 10
      use_bias: true
  name: sequential_4
```

```
keras_version: 2.2.4-tf
```

```
c:\users\aetsl\appdata\local\programs\python\python37\lib\site-packages\tensorf
low_core\python\keras\saving\model_config.py:76: YAMLLoadWarning: calling yaml.
load() without Loader=... is deprecated, as the default Loader is unsafe. Pleas
e read https://msg.pyyaml.org/load (https://msg.pyyaml.org/load) for full detai
ls.
  config = yaml.load(yaml_string)
```

注意：子類模型不可序列化，因為其體系結構由call方法主體中的Python代碼定義。

5.3 保存整個模型

In [18]:
```
model = tf.keras.Sequential([
  layers.Dense(10, activation='softmax', input_shape=(72,)),
  layers.Dense(10, activation='softmax')
])
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_x, train_y, batch_size=32, epochs=5)
# 保存整個模型
model.save('all_model.h5')
# 導入整個模型
model = tf.keras.models.load_model('all_model.h5')
```

```
Epoch 1/5
32/32 [==============================] - 0s 922us/step - loss: 11.5710 - accura
cy: 0.1120
Epoch 2/5
32/32 [==============================] - 0s 961us/step - loss: 11.6205 - accura
cy: 0.1170
Epoch 3/5
32/32 [==============================] - 0s 1ms/step - loss: 11.6548 - accurac
y: 0.1170
Epoch 4/5
32/32 [==============================] - 0s 844us/step - loss: 11.6646 - accura
cy: 0.1170
Epoch 5/5
32/32 [==============================] - 0s 853us/step - loss: 11.6672 - accura
cy: 0.1170
```

6 將keras用於Estimator

Estimator API 用於針對分散式環境訓練模型。它適用於一些行業使用場景，例如用大型資料集進行分散式訓練並匯出模型以用於生產

In [19]:
```
model = tf.keras.Sequential([layers.Dense(10,activation='softmax'),
                             layers.Dense(10,activation='softmax')])

model.compile(optimizer=tf.keras.optimizers.RMSprop(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

estimator = tf.keras.estimator.model_to_estimator(model)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\ASUS_U~1
\AppData\Local\Temp\tmpghqqsazi
INFO:tensorflow:Using the Keras model provided.
INFO:tensorflow:Using config: {'_model_dir': 'C:\\Users\\ASUS_U~1\\AppData\\Loc
al\\Temp\\tmpghqqsazi', '_tf_random_seed': None, '_save_summary_steps': 100, '_
save_checkpoints_steps': None, '_save_checkpoints_secs': 600, '_session_confi
g': allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_ste
p_count_steps': 100, '_train_distribute': None, '_device_fn': None, '_protoco
l': None, '_eval_distribute': None, '_experimental_distribute': None, '_experim
ental_max_worker_delay_secs': None, '_session_creation_timeout_secs': 7200, '_s
ervice': None, '_cluster_spec': ClusterSpec({}), '_task_type': 'worker', '_task
_id': 0, '_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '',
'_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

7 Eager execution

Eager execution是一個動態執行的程式設計環境，它可以立即評估操作。Keras不需要此功能，但它受tf.keras程式支援和對檢查程式和調試有用。

所有的tf.keras模型構建API都與Eager execution相容。儘管可以使用Sequential和函數API，但Eager execution有利於模型子類化和構建自訂層：其要求以代碼形式編寫前向傳遞的API（而不是通過組裝現有層來創建模型的API）。

8 多GPU上運行

tf.keras模型可使用tf.distribute.Strategy在多個GPU上運行 。該API在多個GPU上提供了分散式培訓，幾乎無需更改現有代碼。 當前tf.distribute.MirroredStrategy是唯一受支援的分發策略。MirroredStrategy在單台電腦上使用全縮減進行同步訓練來進行圖內複製。要使用distribute.Strategys，請將優化器產生實體以及模型構建和編譯嵌套在Strategys中.scope()，然後訓練模型。 以下示例tf.keras.Model在單個電腦上的多GPU分配。 首先，在分散式策略範圍內定義一個模型

In [20]:
```
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
    model = tf.keras.Sequential()
    model.add(layers.Dense(16, activation='relu', input_shape=(10,)))
    model.add(layers.Dense(1, activation='sigmoid'))
    optimizer = tf.keras.optimizers.SGD(0.2)
    model.compile(loss='binary_crossentropy', optimizer=optimizer)
model.summary()
```

```
WARNING:tensorflow:There are non-GPU devices in `tf.distribute.Strategy`, not u
sing nccl allreduce.
INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/
task:0/device:CPU:0',)
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 16)                176
_________________________________________________________________
dense_24 (Dense)             (None, 1)                 17
=================================================================
Total params: 193
Trainable params: 193
Non-trainable params: 0
_________________________________________________________________
```

然後像單gpu一樣在資料上訓練模型即可

In [21]:
```
x = np.random.random((1024, 10))
y = np.random.randint(2, size=(1024, 1))
x = tf.cast(x, tf.float32)
dataset = tf.data.Dataset.from_tensor_slices((x, y))
dataset = dataset.shuffle(buffer_size=1024).batch(32)
model.fit(dataset, epochs=1)
```

```
WARNING:tensorflow:From c:\users\asus_ux331\appdata\local\programs\python\pytho
n37\lib\site-packages\tensorflow\python\data\ops\multi_device_iterator_ops.py:6
01: get_next_as_optional (from tensorflow.python.data.ops.iterator_ops) is depr
ecated and will be removed in a future version.
Instructions for updating:
Use `tf.data.Iterator.get_next_as_optional()` instead.
32/32 [==============================] - 0s 1ms/step - loss: 0.6972
```

Out[21]:
```
<tensorflow.python.keras.callbacks.History at 0x1a917611b48>
```