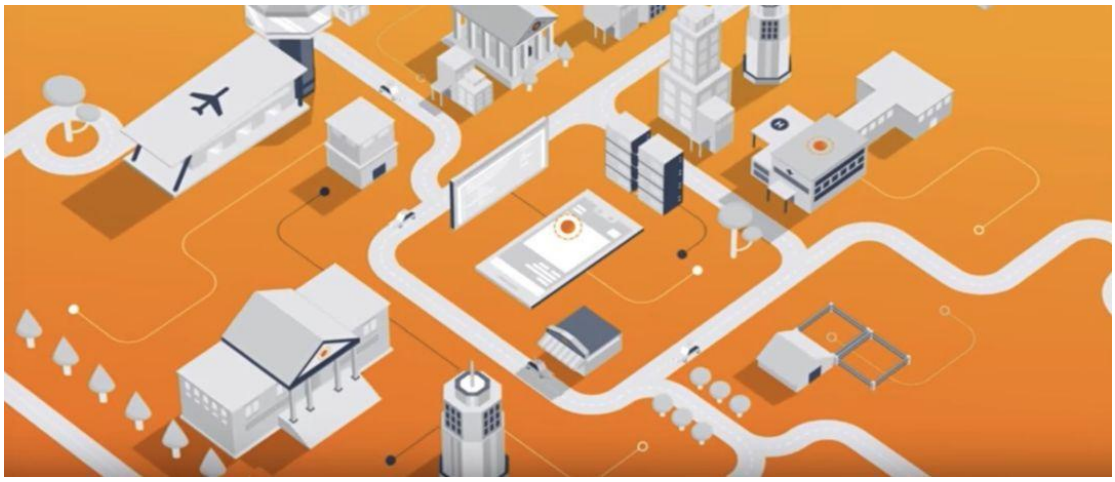
教你如何迁移：将您的代码从 TensorFlow 1 迁移到 TensorFlow 2

（原文链接 -1）
（原文链接-2）

如果你对 TensorFlow 感兴趣并希望开始学习机器学习，我们建议你直接安装对新手更为友好的 TensorFlow 2.X 版本，一键查收安装指南： TensorFlow 2.0 官方安装文档 或 社区内的安装指南。

如果你已经是 TensorFlow 1.x 的用户，我们建议你更新至最新版本，它将为你带来更流畅的机器学习体验。

（一）



- 在 Google Colab 中运行
- 在 GitHub 上查看源代码
- 下载笔记本

重要说明：
这份文档适用于使用低级别 TensorFlow API 的用户。如果您正在使用高级别 API (tf.keras)，可能无需或仅需对您的代码执行很少操作，便可以让代码完全兼容 TensorFlow 2.0。查看您的优化器默认学习速率。

在 TensorFlow 2.0 中，1.X 的代码不经修改也许还可运行（除了 contrib）：

- Contrib

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

但是，这样做无法让您利用到我们对 TensorFlow 2.0 做出的许多改进。这份指南可以帮助您升级代码，让代码更加简洁、性能更好、更容易维护。

自动转换脚本

尝试执行文档中描述的这些变更之前，首先需运行此升级脚本

。

这是将您的代码升级到 TensorFlow 2.0 的第一步。但您的代码不会因此具有 2.0 的特点。您的代码仍然可以使用 `tf.compat.v1` 端点来访问占位符、会话、集合以及 1.x 版本的其他功能。

高层行为变更

如果您使用 `tf.compat.v1.disable_v2_behavior()`让您的代码可以在 TensorFlow 2.0 中工作，那么您仍需要处理全局行为变更。重大变更有：

- *Eager execution*，`v1.enable_eager_execution()`：任何隐式使用 `tf.Graph` 的代码都会执行失败。确保将这个代码打包进 `with tf.Graph().as_default()` 上下文中。
- *资源变量*，`v1.enable_resource_variables()`：一些代码可能会依赖由 TF 参考变量启用的非确定性行为。对资源变量进行写入时，它会处于锁定状态，因此可以提供更直观的一致性保证。
  - 在边缘情形中，启用资源变量可能会改变其中的行为。
  - 启用资源变量可能会创建额外的副本以及有更高的内存使用情况。
  - 可以通过将 `use_resource=False` 传递给 `tf.Variable` 构造器来禁用资源变量。
- *Tensor Shapes*，`v1.enable_v2_tensorshape()`：TF 2.0 可简化张量形状的行为。您可以使用 `t.shape[0]`，而不需要使用 `t.shape[0].value`。这样的变更很小，因此最好立即加以修复。有关示例，请参阅 TensorShape。
- *Control flow*，`v1.enable_control_flow_v2()`：TF 2.0 中的控制流实施已得到简化，因此会有不同的图表征。如有任何问题，请提交错误报告。
- 提交错误报告

让代码成为 2.0 原生代码

这份指南会逐步讲解将 TensorFlow 1.x 代码转换成 TensorFlow 2.0 代码的示例。这一变更让您的代码可以获得性能优化和简化的 API 调用的优势。

在以下各种情况下，模式为：

1. 替换 `v1.Session.run` 调用

每一个 `v1.Session.run` 调用都应该替换为一个 Python 函数。

- `feed_dict` 和 `v1.placeholder` 成为函数参数；

- fetches 成为函数的返回值；
- 在转换期间，即刻执行使用标准 Python 工具（如 pdb）让调试变得简单。

之后添加一个 tf.function 修饰器，这样可以更有效率地在图中运行。如需了解其工作原理的更多内容，请参阅 AutoGraph 指南 。

- AutoGraph 指南

请注意：

- 与 v1.Session.run 不同，tf.function 有固定的返回签名，并且总是返回所有的输出。如果这样会导致性能问题，建议创建两个单独的函数；
- 不需要进行 tf.control_dependencies 或类似的操作：tf.function 会像事先写好一样，按顺序运行。例如，tf.Variable 赋值和 tf.assert 就会自动执行。

2. 使用 Python 对象来追踪变量和损失

在 TF 2.0 中，强烈建议不要使用基于名称的变量追踪。请使用 Python 对象来追踪变量。

请使用 tf.Variable，而不要使用 v1.get_variable。

每一个 v1.variable_scope 应该转换为一个 Python 对象。通常是下列对象中的一个：

- tf.keras.layers.Layer
- tf.keras.Model
- tf.Module

如果您需要聚合变量列表(如 tf.Graph.get_collection(tf.GraphKeys.VARIABLES)，请使用 Layer 和 Model 对象中的.variables 和.trainable_variables 属性。

这些 Layer 和 Model 类会实施多种其他属性，因此不需要全局集合。其 .losses 属性可以替代 tf.GraphKeys.LOSSES 集合。

请参阅 Keras 指南了解详情。

- Keras 指南

警告：

许多 tf.compat.v1 符号会以隐式方式使用全局集合。

3. 升级您的训练循环

使用适合您用例的最高级别 API。推荐使用 tf.keras.Model.fit 构建自己的训练循环。这些高级别函数负责管理许多自己编写训练循环时容易漏掉的低级别细节。例如，这些函数会自动收集正则化损失，并且当调用模型时设置 training=True 参数。

4. 升级您的数据输入流水线

使用 tf.data 数据集进行数据输入。这些对象高效、可表达，与 TensorFlow 有很好的集成。

可以直接传递到 tf.keras.Model.fit 方法。

```
model.fit(dataset, epochs=5)
```

可以直接通过标准 Python 来遍历：

```
for example_batch, label_batch in dataset:
  break
```

5. 迁移 compat.v1 符号

tf.compat.v1 模块含有完整的 TensorFlow 1.x API，且具有其原始的语义。

如果此类转换安全，则 TF2 升级脚本会将符号转换成 2.0 版本中对应的符号，即脚本能够确认行为在 2.0 版本与 1.x 中完全等效（例如，脚本判断两者是同一个函数，因此将 v1.arg_max 重命名为 tf.argmax）。升级脚本运行结束后，会留下一段代码，其中很可能多次出现 compat.v1。建议逐行检查代码，手动将其转换成 2.0 版本中对应的符号（如果有对应的符号，则会在日志中提及）。

模型转换

1. 设置

```
from __future__ import absolute_import, division, print_function,
unicode_literals
import tensorflow as tf

import tensorflow_datasets as tfds
```

2. 低级别变量和运算符的执行

低级别 API 使用方法示例包括：

- 使用变量作用域控制重用
- 使用 v1.get_variable 创建变量。
- 以显式方式访问集合
- 以隐式方式访问集合，方法如下：
  - v1.global_variables
  - v1.losses.get_regularization_loss
- 使用 v1.placeholder 设置图输入
- 使用 Session.run 执行图
- 手动初始化变量

2.1 转换前

使用 TensorFlow 1.x 的代码中的这些模式如下所示。

```
in_a = tf.placeholder(dtype=tf.float32, shape=(2))
in_b = tf.placeholder(dtype=tf.float32, shape=(2))

def forward(x):
  with tf.variable_scope("matmul", reuse=tf.AUTO_REUSE):
    W = tf.get_variable("W", initializer=tf.ones(shape=(2,2)),
                        regularizer=tf.contrib.layers.l2_regularizer(0.04))
    b = tf.get_variable("b", initializer=tf.zeros(shape=(2)))
    return W * x + b

out_a = forward(in_a)
out_b = forward(in_b)

reg_loss=tf.losses.get_regularization_loss(scope="matmul")

with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  outs = sess.run([out_a, out_b, reg_loss],
                  feed_dict={in_a: [1, 0], in_b: [0, 1]})
```

2.2 转换后

在转换后的代码中：

- 变量都是本地 Python 对象
- forward 函数仍然定义计算
- Session.run 的调用转换成对 forward 的调用
- 如果出于性能方面的考虑，也可以添加可选的 tf.function 修饰器
  可以手动计算正则化，无需参考任何全局集合
- **没有会话或占位符**

```
W = tf.Variable(tf.ones(shape=(2,2)), name="W")
b = tf.Variable(tf.zeros(shape=(2)), name="b")

@tf.function
def forward(x):
  return W * x + b
out_a = forward([1,0])
print(out_a)
tf.Tensor(
[[1. 0.]
 [1. 0.]], shape=(2, 2), dtype=float32)
```

```
out_b = forward([0,1])

regularizer = tf.keras.regularizers.l2(0.04)
reg_loss=regularizer(W)
```

3. 基于 tf.layers 的模型

v1.layers 模块用来包含需要依赖 v1.variable_scope 的层函数以定义和重用变量。

3.1 转换前

```
def model(x, training, scope='model'):
  with tf.variable_scope(scope, reuse=tf.AUTO_REUSE):
    x = tf.layers.conv2d(x, 32, 3, activation=tf.nn.relu,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(0.04))
    x = tf.layers.max_pooling2d(x, (2, 2), 1)
    x = tf.layers.flatten(x)
    x = tf.layers.dropout(x, 0.1, training=training)
    x = tf.layers.dense(x, 64, activation=tf.nn.relu)
    x = tf.layers.batch_normalization(x, training=training)
    x = tf.layers.dense(x, 10)
    return x

train_out = model(train_data, training=True)
test_out = model(test_data, training=False)
```

3.2 转换后

- 这一简单的层堆栈可整齐地置于 tf.keras.Sequential 中（如需了解更加复杂的模型，请参阅自定义层和模型，以及函数式 API）
- 这个模型可以追踪变量和正则化损失
- 这个转换过程为一对一进行，因为有从 v1.layers 到 tf.keras.layers 的直接映射

大部分参数保持不变。但要注意区别：

- training 参数运行时，模型将其传递到每个层
- 原始 model 函数的第一个参数（输入 x）会消失。这是因为对象层会将构建模型和调用模型分开

另请注意：

- 如果您使用的是 tf.contrib 初始化器的正则化器，则相比其他正则化器，会有更多的参数变动
- 代码不再写入集合，v1.losses.get_regularization_loss 之类的函数不再返回这些值，因此很可能破坏您的训练循环

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu',
                           kernel_regularizer=tf.keras.regularizers.l2(0.04),
                           input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(10)
])

train_data = tf.ones(shape=(1, 28, 28, 1))
test_data = tf.ones(shape=(1, 28, 28, 1))
train_out = model(train_data, training=True)
print(train_out)
tf.Tensor([[0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]], shape=(1, 10), dtype=float32)
test_out = model(test_data, training=False)
print(test_out)
tf.Tensor(
[[0.08641145 0.07439621 0.08764433 0.08413846 0.13945633 0.10146178
  0.08674773 0.10674053 0.13481253 0.09819068]], shape=(1, 10), dtype=float32
)
# Here are all the trainable variables.
len(model.trainable_variables)
8
# Here is the regularization loss.
model.losses
[<tf.Tensor: shape=(), dtype=float32, numpy=0.07632194>]
```

4. 混合变量和 v1.layers

现有代码经常将低级别 TF 1.x 的变量和运算与高级别的 v1.layers 混合。

4.1 转换前

```
def model(x, training, scope='model'):
  with tf.variable_scope(scope, reuse=tf.AUTO_REUSE):
    W = tf.get_variable(
      "W", dtype=tf.float32,
      initializer=tf.ones(shape=x.shape),
      regularizer=tf.contrib.layers.l2_regularizer(0.04),
      trainable=True)
    if training:
```

```
    x = x + W
  else:
    x = x + W * 0.5
  x = tf.layers.conv2d(x, 32, 3, activation=tf.nn.relu)
  x = tf.layers.max_pooling2d(x, (2, 2), 1)
  x = tf.layers.flatten(x)
  return x

train_out = model(train_data, training=True)
test_out = model(test_data, training=False)
```

4.2 转换后

如要转换这个代码，请遵循之前示例中的层到层的映射模式。

v1.variable_scope 实际上是其自身的层。因此将其重写为 tf.keras.layers.Layer。请参阅此指南以了解详情。

- 指南

通用模式是：

- 在__init__ 中收集层参数
- 在 **build** 中构建变量
- 在 call 中执行计算，然后返回结果

v1.variable_scope 本质上是其自身的层。因此将其重写为 tf.keras.layers.Layer。请参阅此指南以了解详情。

- 指南

```
# Create a custom layer for part of the model
class CustomLayer(tf.keras.layers.Layer):
  def __init__(self, *args, **kwargs):
    super(CustomLayer, self).__init__(*args, **kwargs)

  def build(self, input_shape):
    self.w = self.add_weight(
        shape=input_shape[1:],
        dtype=tf.float32,
        initializer=tf.keras.initializers.ones(),
        regularizer=tf.keras.regularizers.l2(0.02),
        trainable=True)

  # Call method will sometimes get used in graph mode,
  # training will get turned into a tensor
```

```
  @tf.function
  def call(self, inputs, training=None):
    if training:
      return inputs + self.w
    else:
      return inputs + self.w * 0.5
custom_layer = CustomLayer()
print(custom_layer([1]).numpy())
print(custom_layer([1], training=True).numpy())

[1.5]
[2.]

train_data = tf.ones(shape=(1, 28, 28, 1))
test_data = tf.ones(shape=(1, 28, 28, 1))

# Build the model including the custom layer
model = tf.keras.Sequential([
    CustomLayer(input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
])

train_out = model(train_data, training=True)
test_out = model(test_data, training=False)
```

请注意下列事项：

- 子类 Keras 模型和层需要在 v1 图（自动控制依赖项）中以即刻模式运。
  - 将 call()打包进 tf.function()来获得自动图和自动控制依赖项。
- 不要忘记接受一个 call 的 training 参数。
  - 有时是 tf.Tensor。
  - 有时是 Python 布尔值。
- 使用 self.add_weight() 在构造函数或 Model.build 中创建模型变量。
  - 在 Model.build 中，您可以访问输入形状，因此可以创建具有匹配形状的权重。
  - 用 tf.keras.layers.Layer.add_weight 后，Keras 可以追踪变量和正则化损失。
- 不要在您的对象中保留 tf.Tensors。
  - 否则可能会在 tf.function 或即刻上下文中创建，导致产生这些行为不一样的张量。

- 使用 tf.Variable 获得状态，而这些变量在两种上下文中可以一直使用
- tf.Tensors 只适用于中间值。

5. Slim 和 contrib.layers 注意事项

很大一部分旧的 TensorFlow 1.x 代码使用 Slim 代码库，可使用 TensorFlow 1.x 将其打包成 tf.contrib.layers。作为 contrib 的模块，在 TensorFlow 2.0 中将不再可用，即使在 tf.compat.v1 中亦如此。将使用 Slim 的代码转换到 TF 2.0 比转换使用 v1.layers 的代码库涉及的因素更多。事实上，最好先将您的 Slim 代码转换到 v1.layers，然后再转换到 Keras。

- 移除 arg_scopes，所有参数均需要为显式；
- 如要使用，请将 normalizer_fn 和 activation_fn 分离至其自己的层中；
- 可分离的卷积层映射至一个或多个不同的 Keras 层（深度、逐点和可分离 Keras 层）；
- Slim 与 v1.layers 有不同的参数名和默认值；
- 一些参数有不同的尺度；
- 如果您使用 Slim 预训练模型，可以尝试来自 tf.keras.applications 的 Keras 预训练模型或从原始 Slim 代码导出的 TF Hub TF2 SavedModel。
- Slim
  一些 tf.contrib 层没有移至核心 TensorFlow，相反移至 TF add-ons 组件包中。
- TF add-ons 组件包

训练

有许多方法可以将数据喂给 tf.keras 模型。这些方法可以将 Python 生成器和 Numpy 数组作为输入接收。

推荐使用 tf.data 软件包将数据喂给模型，其中包含操作数据的高性能类集合。

如果您使用的仍然是 tf.queue，则现在仅支持将这些当作数据结构，而不能作为输入流水线。

1. 使用数据集

TensorFlow Datasets 软件包 (tfds) 包含将预定义数据集加载成 tf.data.Dataset 对象的实用程序。

- TensorFlow Datasets
- tfds

例如，使用 tfds 加载 MNIST 数据集：

```
datasets, info = tfds.load(name='mnist', with_info=True, as_supervised=True)
mnist_train, mnist_test = datasets['train'], datasets['test']
```

```
Downloading and preparing dataset mnist (11.06 MiB) to /home/kbuilder/tensorflow_datasets/mnist/1.0.0...

WARNING:absl:Dataset mnist is hosted on GCS. It will automatically be downloaded to your
local data directory. If you'd instead prefer to read directly from our public
GCS bucket (recommended if you're running on GCP), you can instead set
data_dir=gs://tfds-data/datasets.

HBox(children=(FloatProgress(value=0.0, description='Dl Completed...', max=19.0, style=ProgressStyle(descripti...

Dataset mnist downloaded and prepared to /home/kbuilder/tensorflow_datasets/mnist/1.0.0. Subsequent calls will reuse this data.
```

然后准备用于训练的数据：

- 重新缩放每个图像；
- 将样本顺序随机排列；
- 收集图像和标签批次。

```
BUFFER_SIZE = 10 # Use a much larger value for real code.
BATCH_SIZE = 64
NUM_EPOCHS = 5

def scale(image, label):
  image = tf.cast(image, tf.float32)
  image /= 255

  return image, label
```

为了使样本简短，将数据集修剪为仅返回 5 个批次：

```
train_data = mnist_train.map(scale).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
test_data = mnist_test.map(scale).batch(BATCH_SIZE)

STEPS_PER_EPOCH = 5

train_data = train_data.take(STEPS_PER_EPOCH)
test_data = test_data.take(STEPS_PER_EPOCH)
image_batch, label_batch = next(iter(train_data))
```

2. 使用 Keras 训练循环

如果您不需要对训练过程进行低级别的控制，推荐使用 Keras 内置的 fit、evaluate 和 predict 方法。无论以何种方式实施（顺序式、函数式或子类式），这些方法都能提供统一的界面用来训练模型。

这些方法的优点有：

- 接受 Numpy 数组、Python 生成器和 tf.data.Datasets；
- 自动应用正则化和激活损失；
- 支持 tf.distribute 来进行多设备训练；
- 支持将任意可调用项作为损失和指标；
- 支持 tf.keras.callbacks.TensorBoard 之类的回调以及自定义回调；
- 性能出色，可自动使用 TensorFlow 图。

下面是使用 Dataset 训练模型的示例（如需了解工作原理详情，请参阅教程。

```
model = tf.keras.Sequential([
  tf.keras.layers.Conv2D(32, 3, activation='relu',
                         kernel_regularizer=tf.keras.regularizers.l2(0.02),
                         input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dropout(0.1),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Dense(10)
])

# Model is the full model w/o custom layers
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.fit(train_data, epochs=NUM_EPOCHS)
loss, acc = model.evaluate(test_data)

print("Loss {}, Accuracy {}".format(loss, acc))

Epoch 1/5
5/5 [==============================] - 1s 135ms/step - loss: 1.5057 - accur
acy: 0.5469
Epoch 2/5
```

```
5/5 [==============================] - 0s 25ms/step - loss: 0.5486 - accur
acy: 0.8844
Epoch 3/5
5/5 [==============================] - 0s 27ms/step - loss: 0.3718 - accura
cy: 0.9375
Epoch 4/5
5/5 [==============================] - 0s 26ms/step - loss: 0.2631 - accura
cy: 0.9656
Epoch 5/5
5/5 [==============================] - 0s 26ms/step - loss: 0.2045 - accur
acy: 0.9875
    5/Unknown - 0s 27ms/step - loss: 1.5769 - accuracy: 0.6562Loss 1.576945734
0240478, Accuracy 0.65625
```

3. 编写自己的循环

如果 Keras 模型训练步骤对您而言有效，但您需要超过此步骤外的更多控制，可以考虑在自己的数据迭代循环中使用 tf.keras.Model.train_on_batch 方法。
请谨记：很多对象可以作为 tf.keras.callbacks.Callback 实施。

这个方法不仅具备上文提及的方法的诸多优势，而且可以让用户控制外循环。

在训练期间，您也可以使用 tf.keras.Model.test_on_batch 或 tf.keras.Model.evaluate 来检查性能。

请注意：
train_on_batch 和 test_on_batch 默认返回单个批次的损失和指标。如果您传入 reset_metrics=False，则会返回累加指标，而且必须记住适当地重置指标累加器。同样请谨记像 AUC 一类的指标需要 reset_metrics=False 来对其准确计算。

为了继续训练上述模型：

```
# Model is the full model w/o custom layers
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

metrics_names = model.metrics_names

for epoch in range(NUM_EPOCHS):
  #Reset the metric accumulators
  model.reset_metrics()
```

```
  for image_batch, label_batch in train_data:
    result = model.train_on_batch(image_batch, label_batch)
    print("train: ",
          "{}: {:.3f}".format(metrics_names[0], result[0]),
          "{}: {:.3f}".format(metrics_names[1], result[1]))
  for image_batch, label_batch in test_data:
    result = model.test_on_batch(image_batch, label_batch,
                                 # return accumulated metrics
                                 reset_metrics=False)
  print("\neval: ",
        "{}: {:.3f}".format(metrics_names[0], result[0]),
        "{}: {:.3f}".format(metrics_names[1], result[1]))
```

```
train:  loss: 0.128 accuracy: 1.000
train:  loss: 0.213 accuracy: 1.000
train:  loss: 0.174 accuracy: 0.969
train:  loss: 0.220 accuracy: 0.984
train:  loss: 0.351 accuracy: 0.906

eval:  loss: 1.542 accuracy: 0.656
train:  loss: 0.076 accuracy: 1.000
train:  loss: 0.118 accuracy: 1.000
train:  loss: 0.102 accuracy: 1.000
train:  loss: 0.125 accuracy: 1.000
train:  loss: 0.188 accuracy: 0.984

eval:  loss: 1.546 accuracy: 0.663
train:  loss: 0.067 accuracy: 1.000
train:  loss: 0.080 accuracy: 1.000
train:  loss: 0.087 accuracy: 1.000
train:  loss: 0.077 accuracy: 1.000
train:  loss: 0.132 accuracy: 0.984

eval:  loss: 1.553 accuracy: 0.694
train:  loss: 0.065 accuracy: 1.000
train:  loss: 0.067 accuracy: 1.000
train:  loss: 0.071 accuracy: 1.000
train:  loss: 0.061 accuracy: 1.000
train:  loss: 0.073 accuracy: 1.000

eval:  loss: 1.516 accuracy: 0.762
train:  loss: 0.058 accuracy: 1.000
train:  loss: 0.061 accuracy: 1.000
```

```
train:  loss: 0.055 accuracy: 1.000
train:  loss: 0.056 accuracy: 1.000
train:  loss: 0.054 accuracy: 1.000

eval:  loss: 1.482 accuracy: 0.800
train:  loss: 0.059 accuracy: 1.000
train:  loss: 0.062 accuracy: 1.000
train:  loss: 0.046 accuracy: 1.000
train:  loss: 0.056 accuracy: 1.000
train:  loss: 0.051 accuracy: 1.000

eval:  loss: 1.451 accuracy: 0.822
```

4. 自定义训练步骤

您可以通过实施自己的训练循环来获得更多的灵活度和控制力。分为三个步骤：

- 遍历 Python 生成器或 `tf.data.Dataset` 来获得一批样本。
- 使用 `tf.GradientTape` 来收集梯度。
- 使用 `tf.keras.optimizers` 中的一个将权重更新应用于模型变量。

请谨记：

- 总是在子类层和模型的 `call` 方法上加入 `training` 参数。
- 确保 `training` 参数设置准确时调用模型。
- 直到模型基于一批数据运行后，模型变量才会存在，具体视使用方法而定。
- 您需要手动处理模型正则化损失这类的对象。

与 v1 相比的简化如下：

- 无需运行变量初始化器。变量创建时已初始化。
- 无需添加手动控制依赖项。即使在 `tf.function` 中，操作也像在即刻模式中一样。

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu',
                           kernel_regularizer=tf.keras.regularizers.l2(0.02),
                           input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(10)
])
```

```
optimizer = tf.keras.optimizers.Adam(0.001)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

@tf.function
def train_step(inputs, labels):
  with tf.GradientTape() as tape:
    predictions = model(inputs, training=True)
    regularization_loss=tf.math.add_n(model.losses)
    pred_loss=loss_fn(labels, predictions)
    total_loss=pred_loss + regularization_loss

  gradients = tape.gradient(total_loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

for epoch in range(NUM_EPOCHS):
  for inputs, labels in train_data:
    train_step(inputs, labels)
  print("Finished epoch", epoch)

Finished epoch 0
Finished epoch 1
Finished epoch 2
Finished epoch 3
Finished epoch 4
```

5. 新版指标和损失

在 TensorFlow 2.0 中，指标和损失是对象。两者都是即刻工作模式，且都位于 `tf.function` 中。

损失对象是可调用型，并且使用 (y_true, y_pred) 作为参数：

```
cce = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
cce([[1, 0]], [[-1.0,3.0]]).numpy()
4.01815
```

指标对象有以下方法：

- `Metric.update_state()` — 添加新的观察结果；
- `Metric.result()` — 依据观察到的值，得到指标的当前结果；
- `Metric.reset_states()` — 清除所有观察结果。

对象本身是可调用型。使用 `update_state` 后，调用会用新的观察结果更新状态，并返回指标的新结果。

您不需要手动初始化指标的变量，而且由于 TensorFlow 2.0 有自动控制依赖项，所以也不需要担心这两点。
下面的代码使用一个指标来跟踪自定义训练循环中观察到的平均损失。

```
# Create the metrics
loss_metric = tf.keras.metrics.Mean(name='train_loss')
accuracy_metric =
tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

@tf.function
def train_step(inputs, labels):
  with tf.GradientTape() as tape:
    predictions = model(inputs, training=True)
    regularization_loss=tf.math.add_n(model.losses)
    pred_loss=loss_fn(labels, predictions)
    total_loss=pred_loss + regularization_loss

  gradients = tape.gradient(total_loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))
  # Update the metrics
  loss_metric.update_state(total_loss)
  accuracy_metric.update_state(labels, predictions)


for epoch in range(NUM_EPOCHS):
  # Reset the metrics
  loss_metric.reset_states()
  accuracy_metric.reset_states()

  for inputs, labels in train_data:
    train_step(inputs, labels)
  # Get the metric results
  mean_loss=loss_metric.result()
  mean_accuracy = accuracy_metric.result()

  print('Epoch: ', epoch)
  print('  loss:     {:.3f}'.format(mean_loss))
  print('  accuracy: {:.3f}'.format(mean_accuracy))

Epoch:  0
  loss:     0.173
  accuracy: 0.997
Epoch:  1
  loss:     0.138
```

```
 accuracy: 1.000
Epoch: 2
 loss:    0.117
 accuracy: 1.000
Epoch: 3
 loss:    0.099
 accuracy: 1.000
Epoch: 4
 loss:    0.093
 accuracy: 0.997
```

6. Keras 优化器

v1.train 中的优化器（如 v1.train.AdamOptimizer 和 v1.train.GradientDescentOptimizer）可以在 tf.keras.optimizers 中找到对应项。

将 v1.train 转换到 keras.optimizers

转换优化器时，请注意以下几点：

- 升级优化器可能会使旧的检查点不兼容。
- 现在，所有的 ε (epsilons) 默认值为 1e-7 而不是 1e-8（大多数情况下可以忽略不计）。
- v1.train.GradientDescentOptimizer 可以用 tf.keras.optimizers.SGD 直接替代。
- 可以通过使用 momentum 参数：tf.keras.optimizers.SGD(..., momentum=...) 将 v1.train.MomentumOptimizer 直接替换成 SGD 优化器。
- v1.train.AdamOptimizer 能够经过转换来使用 tf.keras.optimizers.Adam。beta1 和 beta2 参数已重命名为 beta_1 和 beta_2。
- v1.train.RMSPropOptimizer 可以转换为 tf.keras.optimizers.RMSprop。decay 参数已重命名为 rho。
- v1.train.AdadeltaOptimizer 可以直接转换为 tf.keras.optimizers.Adadelta。
- tf.train.AdagradOptimizer 可以直接转换为 tf.keras.optimizers.Adagrad。
- tf.train.FtrlOptimizer 可以直接转换为 tf.keras.optimizers.Ftrl。accum_name 和 linear_name 参数已移除。
- tf.contrib.AdamaxOptimizer 和 tf.contrib.NadamOptimizer 可以直接转换为 tf.keras.optimizers.Adamax 和 tf.keras.optimizers.Nadam。beta1 和 beta2 参数已重命名为 beta_1 和 beta_2。

一些 tf.keras.optimizers 的全新默认值。

警告：
如果您发现模型的收敛行为发生变化，请检查默认学习速率。
optimizers.SGD、optimizers.Adam 或 optimizers.RMSprop 没有做任何变更。

以下几个优化器的默认学习速率已变更：

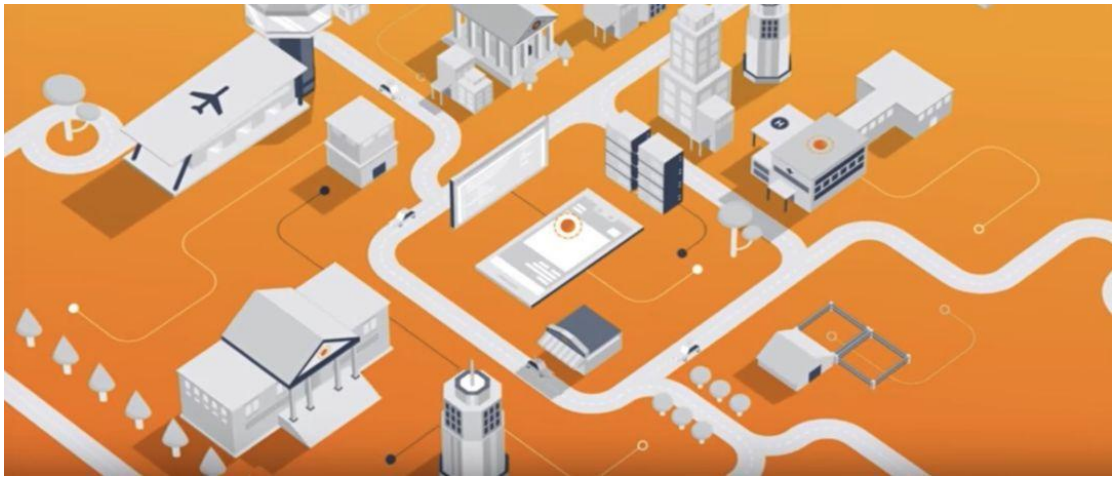
- optimizers.Adagrad 从 0.01 变更到 0.001
- optimizers.Adadelta 从 1.0 变更到 0.001
- optimizers.Adamax 从 0.002 变更到 0.001
- optimizers.Nadam 从 0.002 变更到 0.001

7. TensorBoard

TensorFlow 2.0 包含 tf.summary API 的重大更新，该 API 用于写入摘要数据以在 TensorBoard 中进行可视化。有使用 TF 2.0 API 编写的多份教程可为您提供全新 tf.summary 的总体介绍。其中包含一份 TensorBoard TF2.0 迁移指南。

- 多份教程
- TensorBoard TF2.0 迁移指南

（二）



保存和加载

检查点兼容性

TensorFlow 2.0 使用基于对象的检查点。

- 基于对象的检查点

如果您足够小心，旧版的基于名称的检查点仍然可以加载。如果进行代码转换，可能需要对变量名进行变更，但是也有变通的办法。

最简单的方法是将新模型中的名字和检查点的名字保持一致：

- 您依旧可以为所有变量设置 name 参数。

- Keras 模型也有一个 name 参数，模型将其设置成自有变量的前缀。
- v1.name_scope 函数可以用于设置变量名前缀。这与 tf.variable_scope 有极大的不同。因其只影响名称，不追踪变量和重用。

如果无法适用于您的用例，您可以尝试 v1.train.init_from_checkpoint 函数。该函数带有一个 assignment_map 参数，可以指定从旧名称到新名称的映射。

请注意：
基于对象的检查点可以延迟加载，但基于名称的检查点不能，要求在构建函数时调用所有变量。一些模型只有当您调用 build 或在一个批次的数据上运行模型时才会构建变量。

TensorFlow Estimator 代码库包含一个 转换工具，可以将预制 Estimator 的检查点从 TensorFlow 1.X 升级到 2.0。因此，可以作为如何为类似用例构建工具的示例。

- TensorFlow Estimator 代码库
- 转换工具

已保存模型的兼容性

已保存模型没有太大的兼容性问题。

- TensorFlow 1.x saved_models 可以在 TensorFlow 2.0 中使用。
- 如果 TensorFlow 2.0 能够支持 TensorFlow 1.x 的所有算子，TensorFlow 2.0 saved_models 甚至可以加载 TensorFlow 1.x 中的工作。

Graph.pb 或 Graph.pbtxt

没有直接的方法将原始的 Graph.pb 文件升级到 TensorFlow 2.0。您最好的选择是升级生成文件的代码。
但如果您有一个 *Frozen graph*（tf.Graph 的一种，其中的变量变为常量），那么可以使用 v1.wrap_function 将其转换为 concrete_function：

```
def wrap_frozen_graph(graph_def, inputs, outputs):
  def _imports_graph_def():
    tf.compat.v1.import_graph_def(graph_def, name="")
  wrapped_import = tf.compat.v1.wrap_function(_imports_graph_def, [])
  import_graph = wrapped_import.graph
  return wrapped_import.prune(
      tf.nest.map_structure(import_graph.as_graph_element, inputs),
      tf.nest.map_structure(import_graph.as_graph_element, outputs))
```

此例中有个 *Frozen graph*:

```
path = tf.keras.utils.get_file(
    'inception_v1_2016_08_28_frozen.pb',

    'http://storage.googleapis.com/download.tensorflow.org/models/inception_v1_2016_08_28_frozen.pb.tar.gz',
    untar=True)
```

Downloading data from http://storage.googleapis.com/download.tensorflow.org/models/inception_v1_2016_08_28_frozen.pb.tar.gz
24698880/24695710 [==============================] - 2s 0us/step

加载 tf.GraphDef：

```
graph_def = tf.compat.v1.GraphDef()
loaded = graph_def.ParseFromString(open(path,'rb').read())
```

打包至 concrete_function：

```
inception_func = wrap_frozen_graph(
    graph_def, inputs='input:0',
    outputs='InceptionV1/InceptionV1/Mixed_3b/Branch_1/Conv2d_0a_1x1/Relu:0')
```

向其传递张量，作为输入：

```
input_img = tf.ones([1,224,224,3], dtype=tf.float32)
inception_func(input_img).shape
```

TensorShape([1, 28, 28, 96])

Estimator

使用 Estimator 进行训练

TensorFlow 2.0 支持 Estimator。

当您使用 Estimator 时，可以使用 TensorFlow 1.x 中的 input_fn()、tf.estimator.TrainSpec 和 tf.estimator.EvalSpec。

下面是结合训练和评估规格使用 input_fn 的一个例子。

创建 input_fn 以及训练 / 评估规格：

```
# Define the estimator's input_fn
def input_fn():
  datasets, info = tfds.load(name='mnist', with_info=True, as_supervised=True)
  mnist_train, mnist_test = datasets['train'], datasets['test']

  BUFFER_SIZE = 10000
  BATCH_SIZE = 64

  def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255

    return image, label[..., tf.newaxis]

  train_data = mnist_train.map(scale).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
  return train_data.repeat()

# Define train & eval specs
train_spec = tf.estimator.TrainSpec(input_fn=input_fn,
                                    max_steps=STEPS_PER_EPOCH * NUM_EPOCHS)
eval_spec = tf.estimator.EvalSpec(input_fn=input_fn,
                                  steps=STEPS_PER_EPOCH)
```

使用 Keras 模型定义

TensorFlow 2.0 中构建 Estimator 的方法会有一些不同。我们建议您使用 Keras 定义自己的模型，然后使用 tf.keras.estimator.model_to_estimator 实用程序将模型变为 Estimator。下面的代码展示创建和训练 Estimator 时如何使用这个实用工具。

```
def make_model():
  return tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu',
                           kernel_regularizer=tf.keras.regularizers.l2(0.02),
                           input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.BatchNormalization(),
```

```
    tf.keras.layers.Dense(10)
  ])
```

```
model = make_model()

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

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

tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

```
INFO:tensorflow:Using default config.

INFO:tensorflow:Using default config.

WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmp4q8g
11bh

WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmp4q8g
11bh

INFO:tensorflow:Using the Keras model provided.

INFO:tensorflow:Using the Keras model provided.

WARNING:tensorflow:From /tmpfs/src/tf_docs_env/lib/python3.6/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1635: calling B
aseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops
) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.

WARNING:tensorflow:From /tmpfs/src/tf_docs_env/lib/python3.6/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1635: calling B
aseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops
) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
INFO:tensorflow:Using config: {'_model_dir': '/tmp/tmp4q8g11bh', '_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps': None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None, '_protocol': None, '_eval_distribute': None, '_experimental_distribute': None, '_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs': 7200, '_service': 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}

INFO:tensorflow:Using config: {'_model_dir': '/tmp/tmp4q8g11bh', '_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps': None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None, '_protocol': None, '_eval_distribute': None, '_experimental_distribute': None, '_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs': 7200, '_service': 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}

INFO:tensorflow:Not using Distribute Coordinator.

INFO:tensorflow:Not using Distribute Coordinator.

INFO:tensorflow:Running training and evaluation locally (non-distributed).

INFO:tensorflow:Running training and evaluation locally (non-distributed).

INFO:tensorflow:Start train and evaluate loop. The evaluate will happen after every checkpoint. Checkpoint frequency is determined based on RunConfig arguments: save_checkpoints_steps None or save_checkpoints_secs 600.
```

```
INFO:tensorflow:Start train and evaluate loop. The evaluate will happen after ever
y checkpoint. Checkpoint frequency is determined based on RunConfig argumen
ts: save_checkpoints_steps None or save_checkpoints_secs 600.

WARNING:tensorflow:From /tmpfs/src/tf_docs_env/lib/python3.6/site-
packages/tensorflow_core/python/training/training_util.py:236: Variable.initialize
d_value (from tensorflow.python.ops.variables) is deprecated and will be remove
d in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in eager
 and graph (inside tf.defun) contexts.

WARNING:tensorflow:From /tmpfs/src/tf_docs_env/lib/python3.6/site-
packages/tensorflow_core/python/training/training_util.py:236: Variable.initialize
d_value (from tensorflow.python.ops.variables) is deprecated and will be remove
d in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in eager
 and graph (inside tf.defun) contexts.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Warm-
starting with WarmStartSettings: WarmStartSettings(ckpt_to_initialize_from='/tm
p/tmp4q8g11bh/keras/keras_model.ckpt', vars_to_warm_start='.*', var_name_to_
vocab_info={}, var_name_to_prev_var_name={})

INFO:tensorflow:Warm-
starting with WarmStartSettings: WarmStartSettings(ckpt_to_initialize_from='/tm
p/tmp4q8g11bh/keras/keras_model.ckpt', vars_to_warm_start='.*', var_name_to_
vocab_info={}, var_name_to_prev_var_name={})

INFO:tensorflow:Warm-
starting from: /tmp/tmp4q8g11bh/keras/keras_model.ckpt

INFO:tensorflow:Warm-
starting from: /tmp/tmp4q8g11bh/keras/keras_model.ckpt
```

```
INFO:tensorflow:Warm-starting variables only in TRAINABLE_VARIABLES.

INFO:tensorflow:Warm-starting variables only in TRAINABLE_VARIABLES.

INFO:tensorflow:Warm-started 8 variables.

INFO:tensorflow:Warm-started 8 variables.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Saving checkpoints for 0 into /tmp/tmp4q8g11bh/model.ckpt.

INFO:tensorflow:Saving checkpoints for 0 into /tmp/tmp4q8g11bh/model.ckpt.

INFO:tensorflow:loss = 2.7495928, step = 0

INFO:tensorflow:loss = 2.7495928, step = 0

INFO:tensorflow:Saving checkpoints for 25 into /tmp/tmp4q8g11bh/model.ckpt.

INFO:tensorflow:Saving checkpoints for 25 into /tmp/tmp4q8g11bh/model.ckpt.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.
```

```
INFO:tensorflow:Starting evaluation at 2019-12-21T03:00:31Z

INFO:tensorflow:Starting evaluation at 2019-12-21T03:00:31Z

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from /tmp/tmp4q8g11bh/model.ckpt-25

INFO:tensorflow:Restoring parameters from /tmp/tmp4q8g11bh/model.ckpt-25

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Evaluation [1/5]

INFO:tensorflow:Evaluation [1/5]

INFO:tensorflow:Evaluation [2/5]

INFO:tensorflow:Evaluation [2/5]

INFO:tensorflow:Evaluation [3/5]

INFO:tensorflow:Evaluation [3/5]

INFO:tensorflow:Evaluation [4/5]

INFO:tensorflow:Evaluation [4/5]

INFO:tensorflow:Evaluation [5/5]

INFO:tensorflow:Evaluation [5/5]

INFO:tensorflow:Inference Time : 1.15786s

INFO:tensorflow:Inference Time : 1.15786s
```

```
INFO:tensorflow:Finished evaluation at 2019-12-21-03:00:32

INFO:tensorflow:Finished evaluation at 2019-12-21-03:00:32

INFO:tensorflow:Saving dict for global step 25: accuracy = 0.615625, global_step = 25, loss = 1.4754493

INFO:tensorflow:Saving dict for global step 25: accuracy = 0.615625, global_step = 25, loss = 1.4754493

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 25: /tmp/tmp4q8g11bh/model.ckpt-25

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 25: /tmp/tmp4q8g11bh/model.ckpt-25

INFO:tensorflow:Loss for final step: 0.3878889.

INFO:tensorflow:Loss for final step: 0.3878889.

({'accuracy': 0.615625, 'loss': 1.4754493, 'global_step': 25}, [])
```

使用自定义的 model_fn

如果您已经有一个需要维护的自定义 Estimator `model_fn`，可以转换您的 `model_fn` 以使用 Keras 模型。

然而，由于兼容性问题，自定义 `model_fn` 仍旧会在 1.x 版本图模式中运行。这意味着既没有即刻运行，也没有自动控制依赖项。

只需要很小的变更就能自定义 model_fn

如果您倾向于对现有代码进行最少程度的改动，可以使用 `optimizers` 和 `metrics` 之类的 `tf.compat.v1` 符号，使您的自定义 `model_fn` 能够在 TF 2.0 中正常工作。

在自定义 `model_fn` 中使用 Keras 模型的方式与在自定义训练循环中的使用方式类似：

- 根据 `mode` 参数适当地设置 `training` 阶段。
- 以显式方式将模型的 `trainable_variables` 传递给优化器。

但相对于*自定义循环*，有一些重要的区别：

- 提取损失时，不要使用 `Model.losses`，改为使用 `Model.get_losses_for`。
- 使用 `Model.get_updates_for` 提取模型的更新。

请注意：
“更新”是每一个批次后，需要应用于模型的变更。例如，layers.BatchNormalization 层中均值和方差的移动平均值。

下方的代码将创建一个来自 model_fn 的 Estimator，可说明所有这些问题。

```
def my_model_fn(features, labels, mode):
  model = make_model()

  optimizer = tf.compat.v1.train.AdamOptimizer()
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

  training = (mode == tf.estimator.ModeKeys.TRAIN)
  predictions = model(features, training=training)

  if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

  reg_losses = model.get_losses_for(None) + model.get_losses_for(features)
  total_loss=loss_fn(labels, predictions) + tf.math.add_n(reg_losses)

  accuracy = tf.compat.v1.metrics.accuracy(labels=labels,
                                           predictions=tf.math.argmax(predictions, axis=1),
                                           name='acc_op')

  update_ops = model.get_updates_for(None) + model.get_updates_for(features)
  minimize_op = optimizer.minimize(
      total_loss,
      var_list=model.trainable_variables,
      global_step=tf.compat.v1.train.get_or_create_global_step())
  train_op = tf.group(minimize_op, update_ops)

  return tf.estimator.EstimatorSpec(
    mode=mode,
    predictions=predictions,
    loss=total_loss,
    train_op=train_op, eval_metric_ops={'accuracy': accuracy})

# Create the Estimator & Train
estimator = tf.estimator.Estimator(model_fn=my_model_fn)
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

```
INFO:tensorflow:Using default config.

INFO:tensorflow:Using default config.

WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmp20oz
vzqk

WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmp20oz
vzqk

INFO:tensorflow:Using config: {'_model_dir': '/tmp/tmp20ozvzqk', '_tf_random_se
ed': None, '_save_summary_steps': 100, '_save_checkpoints_steps': None, '_save_
checkpoints_secs': 600, '_session_config': 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, '_protocol': Non
e, '_eval_distribute': None, '_experimental_distribute': None, '_experimental_max_
worker_delay_secs': None, '_session_creation_timeout_secs': 7200, '_service': No
ne, '_cluster_spec': ClusterSpec({}), '_task_type': 'worker', '_task_id': 0, '_global_id_i
n_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chief': True, '_num_ps_replica
s': 0, '_num_worker_replicas': 1}

INFO:tensorflow:Using config: {'_model_dir': '/tmp/tmp20ozvzqk', '_tf_random_se
ed': None, '_save_summary_steps': 100, '_save_checkpoints_steps': None, '_save_
checkpoints_secs': 600, '_session_config': 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, '_protocol': Non
e, '_eval_distribute': None, '_experimental_distribute': None, '_experimental_max_
worker_delay_secs': None, '_session_creation_timeout_secs': 7200, '_service': No
ne, '_cluster_spec': ClusterSpec({}), '_task_type': 'worker', '_task_id': 0, '_global_id_i
n_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chief': True, '_num_ps_replica
s': 0, '_num_worker_replicas': 1}

INFO:tensorflow:Not using Distribute Coordinator.
```

```
INFO:tensorflow:Not using Distribute Coordinator.

INFO:tensorflow:Running training and evaluation locally (non-distributed).

INFO:tensorflow:Running training and evaluation locally (non-distributed).

INFO:tensorflow:Start train and evaluate loop. The evaluate will happen after ever
y checkpoint. Checkpoint frequency is determined based on RunConfig argumen
ts: save_checkpoints_steps None or save_checkpoints_secs 600.

INFO:tensorflow:Start train and evaluate loop. The evaluate will happen after ever
y checkpoint. Checkpoint frequency is determined based on RunConfig argumen
ts: save_checkpoints_steps None or save_checkpoints_secs 600.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Saving checkpoints for 0 into /tmp/tmp20ozvzqk/model.ckpt.

INFO:tensorflow:Saving checkpoints for 0 into /tmp/tmp20ozvzqk/model.ckpt.

INFO:tensorflow:loss = 3.206945, step = 0
```

```
INFO:tensorflow:loss = 3.206945, step = 0

INFO:tensorflow:Saving checkpoints for 25 into /tmp/tmp20ozvzqk/model.ckpt.

INFO:tensorflow:Saving checkpoints for 25 into /tmp/tmp20ozvzqk/model.ckpt.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Starting evaluation at 2019-12-21T03:00:35Z

INFO:tensorflow:Starting evaluation at 2019-12-21T03:00:35Z

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from /tmp/tmp20ozvzqk/model.ckpt-25

INFO:tensorflow:Restoring parameters from /tmp/tmp20ozvzqk/model.ckpt-25

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Evaluation [1/5]

INFO:tensorflow:Evaluation [1/5]

INFO:tensorflow:Evaluation [2/5]

INFO:tensorflow:Evaluation [2/5]

INFO:tensorflow:Evaluation [3/5]
```

```
INFO:tensorflow:Evaluation [3/5]

INFO:tensorflow:Evaluation [4/5]

INFO:tensorflow:Evaluation [4/5]

INFO:tensorflow:Evaluation [5/5]

INFO:tensorflow:Evaluation [5/5]

INFO:tensorflow:Inference Time : 1.22467s

INFO:tensorflow:Inference Time : 1.22467s

INFO:tensorflow:Finished evaluation at 2019-12-21-03:00:37

INFO:tensorflow:Finished evaluation at 2019-12-21-03:00:37

INFO:tensorflow:Saving dict for global step 25: accuracy = 0.509375, global_step
= 25, loss = 1.4795268

INFO:tensorflow:Saving dict for global step 25: accuracy = 0.509375, global_step
= 25, loss = 1.4795268

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 25: /tmp/tmp
20ozvzqk/model.ckpt-25

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 25: /tmp/tmp
20ozvzqk/model.ckpt-25

INFO:tensorflow:Loss for final step: 0.6385002.

INFO:tensorflow:Loss for final step: 0.6385002.

({'accuracy': 0.509375, 'loss': 1.4795268, 'global_step': 25}, [])
```

使用 TF 2.0 符号的自定义 model_fn

如果您想摆脱所有 TF 1.x 符号，然后将自定义的 model_fn 升级到原生 TF 2.0，您需要将 optimizers 和 metric 更新为 tf.keras.optimizers 和 tf.keras.metrics。

在自定义的 model_fn 中，除了以上 *变更*，还需要进行更多升级：

- 请使用 tf.keras.optimizers，而不要使用 v1.train.Optimizer。
- 以显式方式将模型的 trainable_variables 传递到 tf.keras.optimizers。
- 如要计算 train_op/minimize_op，

    - 如果损失是标量损失 Tensor（非可调用型），请使用 Optimizer.get_updates()。返回列表中的第一个元素是所需的 train_op/minimize_op。
    - 如果损失是可调用型（比如一个函数），请使用 Optimizer.minimize()来获取 train_op/minimize_op。
- 评估时，请使用 tf.keras.metrics，而不要使用 tf.compat.v1.metrics。

对于上述的 my_model_fn 示例，使用 2.0 符号的迁移后代码显示为：

```
def my_model_fn(features, labels, mode):
  model = make_model()

  training = (mode == tf.estimator.ModeKeys.TRAIN)
  loss_obj = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
  predictions = model(features, training=training)

  # Get both the unconditional losses (the None part)
  # and the input-conditional losses (the features part).
  reg_losses = model.get_losses_for(None) + model.get_losses_for(features)
  total_loss=loss_obj(labels, predictions) + tf.math.add_n(reg_losses)

  # Upgrade to tf.keras.metrics.
  accuracy_obj = tf.keras.metrics.Accuracy(name='acc_obj')
  accuracy = accuracy_obj.update_state(
      y_true=labels, y_pred=tf.math.argmax(predictions, axis=1))

  train_op = None
  if training:
    # Upgrade to tf.keras.optimizers.
    optimizer = tf.keras.optimizers.Adam()
    # Manually assign tf.compat.v1.global_step variable to optimizer.iterations
    # to make tf.compat.v1.train.global_step increased correctly.
    # This assignment is a must for any `tf.train.SessionRunHook` specified in
    # estimator, as SessionRunHooks rely on global step.
    optimizer.iterations = tf.compat.v1.train.get_or_create_global_step()
    # Get both the unconditional updates (the None part)
    # and the input-conditional updates (the features part).
    update_ops = model.get_updates_for(None) +
model.get_updates_for(features)
    # Compute the minimize_op.
    minimize_op = optimizer.get_updates(
        total_loss,
        model.trainable_variables)[0]
```

```
    train_op = tf.group(minimize_op, *update_ops)

  return tf.estimator.EstimatorSpec(
    mode=mode,
    predictions=predictions,
    loss=total_loss,
    train_op=train_op,
    eval_metric_ops={'Accuracy': accuracy_obj})

# Create the Estimator & Train.
estimator = tf.estimator.Estimator(model_fn=my_model_fn)
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

```
INFO:tensorflow:Using default config.

INFO:tensorflow:Using default config.

WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmpeltbj_
0v

WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmpeltbj_
0v

INFO:tensorflow:Using config: {'_model_dir': '/tmp/tmpeltbj_0v', '_tf_random_see
d': None, '_save_summary_steps': 100, '_save_checkpoints_steps': None, '_save_c
heckpoints_secs': 600, '_session_config': 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, '_protocol': Non
e, '_eval_distribute': None, '_experimental_distribute': None, '_experimental_max_
worker_delay_secs': None, '_session_creation_timeout_secs': 7200, '_service': No
ne, '_cluster_spec': ClusterSpec({}), '_task_type': 'worker', '_task_id': 0, '_global_id_i
n_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chief': True, '_num_ps_replica
s': 0, '_num_worker_replicas': 1}

INFO:tensorflow:Using config: {'_model_dir': '/tmp/tmpeltbj_0v', '_tf_random_see
d': None, '_save_summary_steps': 100, '_save_checkpoints_steps': None, '_save_c
heckpoints_secs': 600, '_session_config': allow_soft_placement: true
graph_options {
```

```
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None, '_protocol': None, '_eval_distribute': None, '_experimental_distribute': None, '_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs': 7200, '_service': 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}

INFO:tensorflow:Not using Distribute Coordinator.

INFO:tensorflow:Not using Distribute Coordinator.

INFO:tensorflow:Running training and evaluation locally (non-distributed).

INFO:tensorflow:Running training and evaluation locally (non-distributed).

INFO:tensorflow:Start train and evaluate loop. The evaluate will happen after every checkpoint. Checkpoint frequency is determined based on RunConfig arguments: save_checkpoints_steps None or save_checkpoints_secs 600.

INFO:tensorflow:Start train and evaluate loop. The evaluate will happen after every checkpoint. Checkpoint frequency is determined based on RunConfig arguments: save_checkpoints_steps None or save_checkpoints_secs 600.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.
```

```
INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Saving checkpoints for 0 into /tmp/tmpeltbj_0v/model.ckpt.

INFO:tensorflow:Saving checkpoints for 0 into /tmp/tmpeltbj_0v/model.ckpt.

INFO:tensorflow:loss = 2.9231493, step = 0

INFO:tensorflow:loss = 2.9231493, step = 0

INFO:tensorflow:Saving checkpoints for 25 into /tmp/tmpeltbj_0v/model.ckpt.

INFO:tensorflow:Saving checkpoints for 25 into /tmp/tmpeltbj_0v/model.ckpt.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Starting evaluation at 2019-12-21T03:00:40Z

INFO:tensorflow:Starting evaluation at 2019-12-21T03:00:40Z

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from /tmp/tmpeltbj_0v/model.ckpt-25

INFO:tensorflow:Restoring parameters from /tmp/tmpeltbj_0v/model.ckpt-25

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.
```

```
INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Evaluation [1/5]

INFO:tensorflow:Evaluation [1/5]

INFO:tensorflow:Evaluation [2/5]

INFO:tensorflow:Evaluation [2/5]

INFO:tensorflow:Evaluation [3/5]

INFO:tensorflow:Evaluation [3/5]

INFO:tensorflow:Evaluation [4/5]

INFO:tensorflow:Evaluation [4/5]

INFO:tensorflow:Evaluation [5/5]

INFO:tensorflow:Evaluation [5/5]

INFO:tensorflow:Inference Time : 1.13835s

INFO:tensorflow:Inference Time : 1.13835s

INFO:tensorflow:Finished evaluation at 2019-12-21-03:00:41

INFO:tensorflow:Finished evaluation at 2019-12-21-03:00:41

INFO:tensorflow:Saving dict for global step 25: Accuracy = 0.728125, global_step
= 25, loss = 1.6920828

INFO:tensorflow:Saving dict for global step 25: Accuracy = 0.728125, global_step
= 25, loss = 1.6920828

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 25: /tmp/tmp
eltbj_0v/model.ckpt-25

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 25: /tmp/tmp
eltbj_0v/model.ckpt-25
```

```
INFO:tensorflow:Loss for final step: 0.43470243.

INFO:tensorflow:Loss for final step: 0.43470243.

({'Accuracy': 0.728125, 'loss': 1.6920828, 'global_step': 25}, [])
```

预制 Estimator

TensorFlow 2.0 API 依然支持 tf.estimator.DNN*、tf.estimator.Linear*和 tf.estimator.DNNLinearCombined*系列中的预制 Estimator，但某些参数已变更：

1. input_layer_partitioner：已在 2.0 中删除。
2. loss_reduction：已升级为 tf.keras.losses.Reduction，而不是 tf.compat.v1.losses.Reduction。其默认值也从 tf.compat.v1.losses.Reduction.SUM 变更为 tf.keras.losses.Reduction.SUM_OVER_BATCH_SIZE。
3. optimizer、dnn_optimizer 和 linear_optimizer：此类参数已更新为 tf.keras.optimizers，而不是 tf.compat.v1.train.Optimizer。

- 预制 Estimator

如要迁移上述变更：

1. input_layer_partitioner 无需迁移，因为在 TF 2.0 中分布策略会自动对其处理。
2. 对于 loss_reduction，请查看 tf.keras.losses.Reduction 获取支持的选项。
3. 对于 optimizer 参数，如果您没有传入 optimizer、dnn_optimizer 或 linear_optimizer 参数，又或您已经在代码中将 optimizer 的参数指定为 string，那么您无需做任何变更。系统会默认使用 tf.keras.optimizers。否则，您需要将其从 tf.compat.v1.train.Optimizer 更新为相应的 tf.keras.optimizers。

检查点转换器

因为 tf.keras.optimizers 会生成不同的参数组并保存到检查点，所以迁移至 keras.optimizers 会破坏使用 TF 1.x 保存的检查点。为了让旧的检查点在您迁移至 TF 2.0 后仍然可用，请尝试 检查点转换器工具。

- 检查点转换器工具

```
! curl -O
https://raw.githubusercontent.com/tensorflow/estimator/master/tensorflow_es
timator/python/estimator/tools/checkpoint_converter.py
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 15371  100 15371    0     0  37674      0 --:--:-- --:--:-- --:--:-- 37581
```

此工具有内置的帮助：

```
! python checkpoint_converter.py -h
```

```
2019-12-
21 03:00:42.978036: I tensorflow/stream_executor/platform/default/dso_loader.
cc:44] Successfully opened dynamic library libnvinfer.so.6
2019-12-
21 03:00:42.978321: W tensorflow/stream_executor/platform/default/dso_loader
.cc:55] Could not load dynamic library 'libnvinfer_plugin.so.6'; dlerror: libnvinfer_pl
ugin.so.6: cannot open shared object file: No such file or directory
2019-12-
21 03:00:42.978341: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:30] Can
not dlopen some TensorRT libraries. If you would like to use Nvidia GPU with Tens
orRT, please make sure the missing libraries mentioned above are installed prope
rly.
usage: checkpoint_converter.py [-h]
                               {dnn,linear,combined} source_checkpoint
                               source_graph target_checkpoint

positional arguments:
  {dnn,linear,combined}
                        The type of estimator to be converted. So far, the
                        checkpoint converter only supports Canned Estimator.
                        So the allowed types include linear, dnn and combined.
  source_checkpoint     Path to source checkpoint file to be read in.
  source_graph          Path to source graph file to be read in.
  target_checkpoint     Path to checkpoint file to be written out.

optional arguments:
  -h, --help            show this help message and exit
```

TensorShape

这个类已简化为保留 `int`， 而不是 `tf.compat.v1.Dimension` 对象。因此不需要调用 `.value()` 来获得 `int`。

仍然可以从 `tf.TensorShape.dims` 中访问单独的 `tf.compat.v1.Dimension` 对象。

以下所示为 TensorFlow 1.x 和 TensorFlow 2.0 之间的区别。

```
# Create a shape and choose an index
i = 0
```

```
shape = tf.TensorShape([16, None, 256])
shape
```

TensorShape([16, None, 256])

如果 TF 1.x 中您的代码是这样：

```
value = shape[i].value
```

请在 TF 2.0 中改为：

```
value = shape[i]
value
```

16

如果 TF 1.x 中您的代码是这样：

```
for dim in shape:
    value = dim.value
    print(value)
```

请在 TF 2.0 中改为：

```
for value in shape:
  print(value)
```

16
None
256

如果 TF 1.x 中您的代码是这样（或使用任何其他维度方法）：

```
dim = shape[i]
dim.assert_is_compatible_with(other_dim)
```

请在 TF 2.0 中改为：

```
other_dim = 16
Dimension = tf.compat.v1.Dimension
```

```
if shape.rank is None:
  dim = Dimension(None)
else:
  dim = shape.dims[i]
dim.is_compatible_with(other_dim) # or any other dimension method
```

```
True
```

```
shape = tf.TensorShape(None)

if shape:
  dim = shape.dims[i]
  dim.is_compatible_with(other_dim) # or any other dimension method
```

如果等级已知，则 tf.TensorShape 的布尔值是 True，否则是 False。

```
print(bool(tf.TensorShape([])))    # Scalar
print(bool(tf.TensorShape([0])))   # 0-length vector
print(bool(tf.TensorShape([1])))   # 1-length vector
print(bool(tf.TensorShape([None]))) # Unknown-length vector
print(bool(tf.TensorShape([1, 10, 100])))     # 3D tensor
print(bool(tf.TensorShape([None, None, None]))) # 3D tensor with no known
dimensions
print()
print(bool(tf.TensorShape(None))) # A tensor with unknown rank.
```

```
True
True
True
True
True
True

False
```

其他变更

- 移除 tf.colocate_with：TensorFlow 的设备布局算法已极大改善。因此不再需要。如果移除会导致性能下降，请提交错误报告。
- 使用来自 tf.config 的对应函数替换 v1.ConfigProto 的用法。
- 提交错误报告

结论

整个过程是：

1. 运行升级脚本。
2. 移除 contrib 符号。
3. 将您的模型切换为面向对象的风格 (Keras)。
4. 尽可能使用 tf.keras 或 tf.estimator 训练和评估循环。
5. 否则，请使用自定义循环，但注意避免会话和集合。

将代码转换为具有 TensorFlow 2.0 特点需要花费一点点功夫，但每一处变更都能带来：

- 代码更短小精炼。
- 逻辑更清晰简明。
- 调试更加方便。

以上教程链接可供微信收藏：

[将您的代码从 TensorFlow 1 迁移到 TensorFlow 2 （一）]
[将您的代码从 TensorFlow 1 迁移到 TensorFlow 2 （二）]

了解如何迁移之后，你可以通过以下方式继续学习：

- **阅读 TensorFlow 文件夹中的其他学习资料：**

  如果想回顾机器学习的基础知识，可以在文件夹里查阅《开始你的 TensorFlow 之旅：【中文教程】机器学习从零到一》，了解基本的计算机视觉概念，卷积神经网络以及其应用和工作原理，自己上手体验机器学习的乐趣。

  如果你对 TensorFlow 已经有初步了解，想要进一步深入学习，就不能错过 TensorFlow 2.2 的重大更新。对此，我们准备了《 全面了解 TensorFlow 2.2 》向你介绍最新特性。

  如果你已经在使用 TensorFlow 2.x ，在使用的过程中遇到难题、感到困惑，希望了解其他开发者是否遇到同样的问题，欢迎阅读《 大神为你码上作答 | 10 个学好 TensorFlow 2.2 的理由 》。通过 TensorFlow 新推出的问答栏目“码上作答”，谷歌开发者专家彭靖田分享了他对 TensorFlow 2.2 的独家学习经验和使用心得，并接受各位开发者的提问。

  同时，配合官网阅读，体验更佳：**https://tensorflow.google.cn/**

- **你还可以加入 TFUG 社区，认识更多优秀开发者，在社区中进步。**
  TFUG，欢迎你的加入！

我们为专业的 TensorFlow 开发者提供正式认证和证书，它不仅能够证明你的学习能力，同时也助力你的职业发展点亮 LinkedIn 技能。

- **关注 TensorFlow 官方微信公众号，回复“认证”，即可获得《 TensorFlow 开发者认证候选人手册》，助你在机器学习道路上更进一步：**

期待你顺利迈出学习 TensorFlow 的第一步！