

AI学习笔记--Tensorflow--TF mode save and load

模型可以在训练期间和训练完成后进行保存。这意味着模型可以从任意中断中恢复，并避免耗费比较长的时间在训练上。保存也意味着您可以共享您的模型，而其他人可以通过您的模型来重新创建工作。在发布研究模型和技术时，大多数机器学习从业者分享：

- 用于创建模型的代码
- 模型训练的权重 (weight) 和参数 (parameters) 。

共享数据有助于其他人了解模型的工作原理，并使用新数据自行尝试。

注意：小心不受信任的代码——Tensorflow 模型是代码。有关详细信息，请参阅 [安全使用Tensorflow](#)。

保存 Tensorflow 的模型有许多方法——具体取决于您使用的 API。本指南使用 `tf.keras`，一个高级 API 用于在 Tensorflow 中构建和训练模型。有关其他方法的实现，请参阅 [TensorFlow 保存和恢复指南](#)或[保存到 eager](#)。

- 配置

安装并导入

安装并导入Tensorflow和依赖项

```
pip install -q pyyaml h5py # 需要以 HDF5 格式保存模型
```

新建一个训练模型：

```
from __future__ import absolute_import, division, print_function,
unicode_literals

import os

import tensorflow as tf
from tensorflow import keras

print(tf.version.VERSION)

# 定义一个简单的序列模型
def create_model():
    model = tf.keras.models.Sequential([
        keras.layers.Dense(512, activation='relu', input_shape=(784,)),
        keras.layers.Dropout(0.2),
        keras.layers.Dense(10, activation='softmax')
    ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```

return model

(train_images, train_labels), (test_images, test_labels) =
tf.keras.datasets.mnist.load_data()

train_labels = train_labels[:1000]
test_labels = test_labels[:1000]

train_images = train_images[:1000].reshape(-1, 28 * 28) / 255.0
test_images = test_images[:1000].reshape(-1, 28 * 28) / 255.0


# 创建一个基本的模型实例
model = create_model()
# 显示模型的结构
model.summary()

checkpoint_path = "training_1/cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

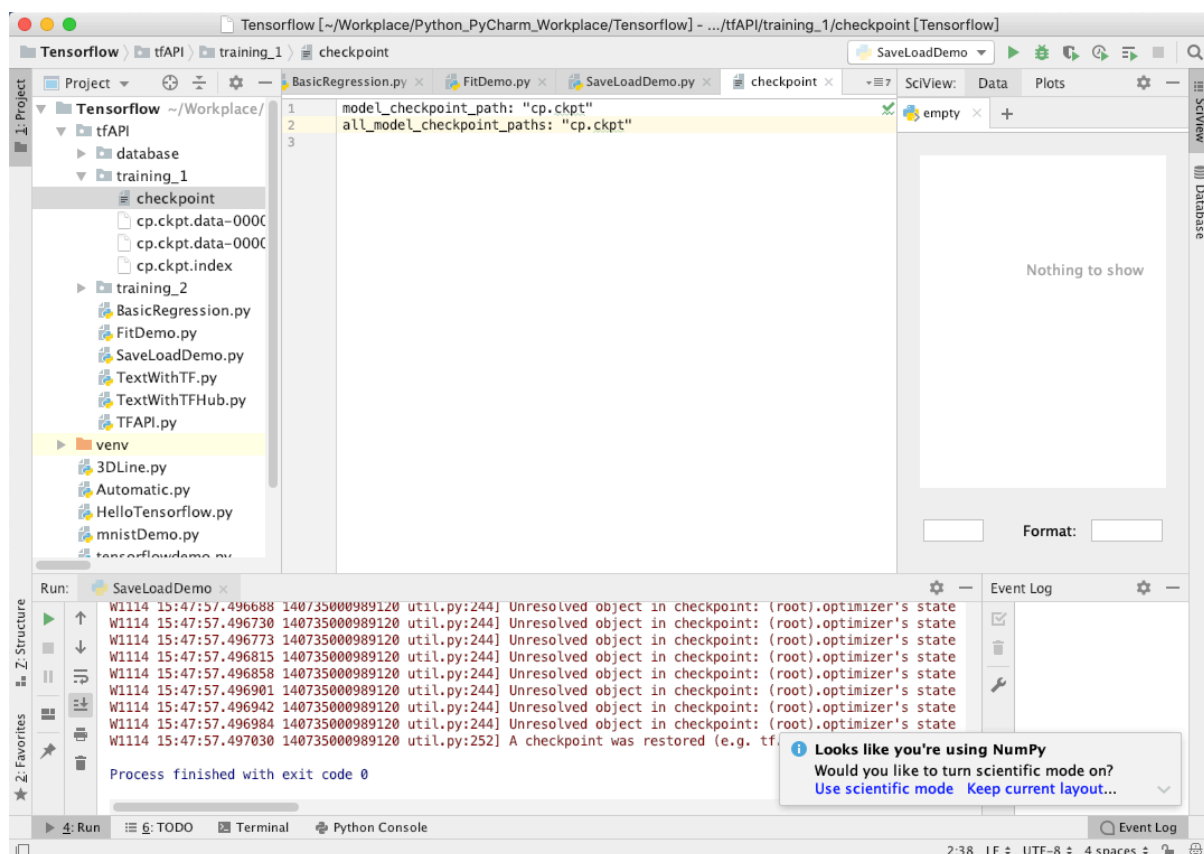
# 创建一个保存模型权重的回调
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                  save_weights_only=True,
                                                  verbose=1)

# 使用新的回调训练模型
model.fit(train_images,
          train_labels,
          epochs=10,
          validation_data=(test_images, test_labels),
          callbacks=[cp_callback]) # 通过回调训练


# 这可能会生成与保存优化程序状态相关的警告。
# 这些警告（以及整个笔记本中的类似警告）是防止过时使用，可以忽略。

```

执行完成后，会在目录生成相应的权值记录文件：



并且打印最终模型的结果：

```
1000/1000 - 0s - loss: 2.4415 - acc: 0.0460
Untrained model, accuracy: 4.60%
1000/1000 - 0s - loss: 0.4256 - acc: 0.8640
Restored model, accuracy: 86.40%
```

上述代码将权重存储到 checkpoint—— 格式化文件的集合中，这些文件仅包含二进制格式的训练权重。Checkpoints 包含： * 一个或多个包含模型权重的分片。 * 索引文件，指示哪些权重存储在哪个分片中。

如果你只在一台机器上训练一个模型，你将有一个带有后缀的碎片：.data-00000-of-00001

创建一个新的未经训练的模型。仅恢复模型的权重时，必须具有与原始模型具有相同网络结构的模型。由于模型具有相同的结构，您可以共享权重，尽管它是模型的不同实例。现在重建一个新的未经训练的模型，并在测试集上进行评估。未经训练的模型将在机会水平（chance levels）上执行（准确度约为10%）：

```
# 这可能会生成与保存优化程序状态相关的警告。
# 这些警告（以及整个笔记本中的类似警告）是防止过时使用，可以忽略。

# 创建一个基本模型实例
model = create_model()

# 评估模型
loss, acc = model.evaluate(test_images, test_labels, verbose=2)
```

```

print("Untrained model, accuracy: {:.2f}%".format(100*acc))

# 加载权重
model.load_weights(checkpoint_path)

# 重新评估模型
loss, acc = model.evaluate(test_images, test_labels, verbose=2)
print("Restored model, accuracy: {:.2f}%".format(100*acc))

# 在文件名中包含 epoch (使用 `str.format`)
checkpoint_path = "training_2/cp-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

# 创建一个回调, 每 5 个 epochs 保存模型的权重
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    verbose=1,
    save_weights_only=True,
    period=5)

# 创建一个新的模型实例
model = create_model()

# 使用 `checkpoint_path` 格式保存权重
model.save_weights(checkpoint_path.format(epoch=0))

# 使用新的回调*训练*模型
model.fit(train_images,
          train_labels,
          epochs=50,
          callbacks=[cp_callback],
          validation_data=(test_images, test_labels),
          verbose=0)

```

注意: 默认的 tensorflow 格式仅保存最近的5个 checkpoint 。

如果要进行测试, 请重置模型并加载最新的 checkpoint :

手动保存权重

您将了解如何将权重加载到模型中。使用 Model.save_weights 方法手动保存它们同样简单。默认情况下, tf.keras 和 save_weights 特别使用 TensorFlow checkpoints 格式 .ckpt 扩展名和 (保存在 HDF5 扩展名为 .h5 保存并序列化模型):

```

# 保存权重
model.save_weights('./checkpoints/my_checkpoint')

# 创建模型实例
model = create_model()

```

```
# Restore the weights
model.load_weights('./checkpoints/my_checkpoint')

# Evaluate the model
loss, acc = model.evaluate(test_images, test_labels, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

- 将模型保存为HDF5文件

Keras 可以使用 HDF5 标准提供基本保存格式。出于我们的目的，可以将保存的模型视为单个二进制blob：

```
# 将整个模型保存为HDF5文件
model.save('my_model.h5')
```

| | |
|--------------------|----|
| ▶ training_2 | 11 |
| BasicRegression.py | 12 |
| FitDemo.py | 13 |
| my_model.h5 | 14 |
| SaveLoadDemo.py | 15 |
| TextWithTF.py | 16 |
| | 17 |
| | 18 |

需要导入则使用 load 方法：

```
# 重新创建完全相同的模型，包括其权重和优化程序
new_model = keras.models.load_model('my_model.h5')

# 显示网络结构
new_model.summary()
```

这项技术可以保存一切：

- 权重
- 模型配置(结构)
- 优化器配置

Keras 通过检查网络结构来保存模型。目前，它无法保存 Tensorflow 优化器（调用自 tf.train）。使用这些的时候，您需要在加载后重新编译模型，否则您将失去优化器的状态。

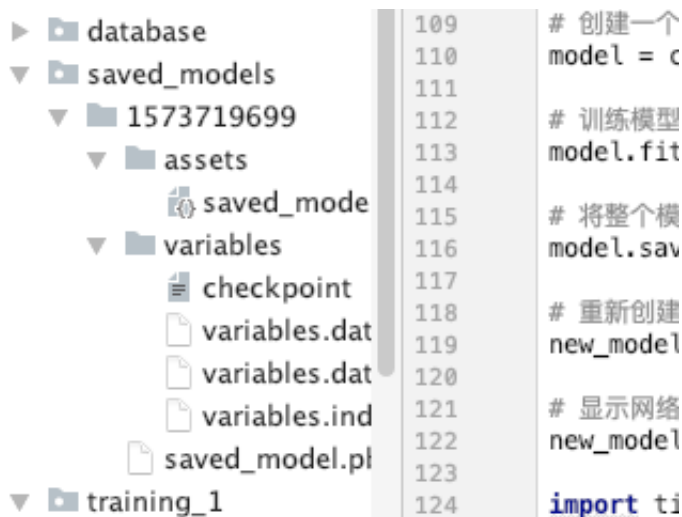
- 通过 saved_model 保存

注意：这种保存 tf.keras 模型的方法是实验性的，在将来的版本中可能有所改变。

创建一个 saved_model，并将其放在带有 tf.keras.experimental.export_saved_model 的带时间戳的目录中：

```
import time
saved_model_path = "./saved_models/{}".format(int(time.time()))

tf.keras.experimental.export_saved_model(model, saved_model_path)
saved_model_path
```



```
109 # 创建一个
110 model = c
111
112 # 训练模型
113 model.fit
114
115 # 将整个模
116 model.sav
117
118 # 重新创建
119 new_model
120
121 # 显示网络
122 new_model
123
124 import ti
```

从保存的模型重新加载新的 Keras 模型：

```
# load save 模型
new_model = tf.keras.experimental.load_from_saved_model(saved_model_path)

# 显示网络结构
new_model.summary()

# 必须在评估之前编译模型。
# 如果仅部署已保存的模型，则不需要此步骤。

new_model.compile(optimizer=model.optimizer, # 保留已加载的优化程序
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

# 评估已恢复的模型
loss, acc = new_model.evaluate(test_images, test_labels, verbose=2)
print("Restored model by save model , accuracy:
{:5.2f}%".format(100*acc))
```

结果：

```
-----
1000/1000 - 0s - loss: 0.4190 - acc: 0.8600
Restored model by save model , accuracy: 86.00%
```

