简介

- 管理TF程序（tf.graph）和TF运行时（tf.Session），而不是依靠Estimator来管理它们
- 使用tf.Session来运行TF指令
- 在此低级别环境中使用高级别组件（数据集、层、特征列）
- 构建自己的训练循环，而不是使用Estimator提供的训练循环

设置

- 基本环境（略）

张量值

- TF中的核心数据单位是张量（tensor）
- 一个张量是由一组形成阵列（array，任意维数）的原始值组成

```
3. # 0阶张量（标量），shape []，

[1., 2., 3.] # 1阶张量（向量）， shape [3]

[[1., 2., 3.], [4., 5., 6.]] # 2阶张量（矩阵）， shape [2, 3]

[[[1., 2., 3.]], [[7., 8., 9.]]] # 3阶张量， shape [2, 1, 3]
```

- TensorFlow 使用 numpy 阵列来表示张量值

TensorFlow核心演示

- TF核心程序可以看做两个互相独立的部分组成：
  - 构建计算图（tf.graph）
  - 运行计算图（tf.Session）

- 图（Graph）
- 计算图是一系列TF指令
- 图由两种类型的对象组成：
  - 指令（Operations）：图的节点。指令说明的是消耗和生成张量的计算
  - 张量：图的边。代表流经图的值，大多数TF函数返回tf.Tensors
- 提示：tf.Tensors 不具有值，它们只是计算图中元素的手柄

- TensorBoard：可视化计算图
- 首先将计算图保存为摘要文件：

```
writer = tf.summary.FileWriter('.')
writer.add_graph(tf.get_default_graph())
```

- 会生成一个event文件
- 启动TensorBoard：

```
tensorboard --logdir .
```

会话（Session）

- 实例化一个 tf.Session 对象，即会话
- 会话会封装 TensorFlow 运行时的状态，并运行 TensorFlow 指令
- 如果说 tf.Graph 像一个 .py 文件，那么 tf.Session 就像一个可执行的 python
- 部分 TensorFlow 函数会返回 tf.Operations，而不是 tf.Tensors
- 对指令调用 run 的结果是 None

供给（Feeding）

- 图可以参数化以便接受外部输入，也称为占位符（placeholders）
- 使用 run 方法的 feed_dict 参数来为占位符提供真正的值

数据集

- 占位符适用于简单的实验，但数据集是将数据流式传输到模型的首选方法
- 要从数据集中获取可运行的 tf.Tensor，必须先将其转换成 tf.data.Iterator
- 然后调用迭代器 (Iterator) 的 get_next 方法
- 创建迭代器的最简单的方式是采用 make_one_shot_iterator 方法：

```
my_data = [
    [0, 1,],
    [2, 3,],
    [4, 5,],
    [6, 7,],
]
slices = tf.data.Dataset.from_tensor_slices(my_data)
next_item = slices.make_one_shot_iterator().get_next()
```

层（Layers）

- 层将变量和作用于它们的指令打包在一起
- 例如，密集连接层会对每个输出值对应的所有输入值执行加权和，并应用可选择的激活函数。连接权重和偏差由层对象管理。

- 创建层

```
x = tf.placeholder(tf.float32, shape=[None, 3])
linear_model = tf.layers.Dense(units=1)
y = linear_model(x)
```

- 初始化层

```
init = tf.global_variables_initializer() # 初始化所有全局变量
sess.run(init)
```

- 执行层

```
print(sess.run(y, {x: [[1, 2, 3],[4, 5, 6]]}))
```

特征列

- 使用特征列进行实验的最简单的方法是使用 tf.feature_column.input_layer 函数

训练

- 损失：均方误差（回归问题）
- tf.losses 模块提供了一系列常用的损失函数
- 训练：TensorFlow 提供了优化器来执行标准的优化算法

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)

linear_model = tf.layers.Dense(units=1)

y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)

optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
  print(loss_value)

print(sess.run(y_pred))
```

更新时间：2018年05月08日10:07:37