数据集快速入门

- tf.data 模块：加载数据、操作数据并通过管道将数据传送到Estimator模型中。
- 两个示例：
  - 从 Numpy 数组中读取内存中的数据
  - 从 csv 文件中读取行

1、基本输入

- 从数组中提取切片
- 鸢尾花分类任务中，train_input_fn可以通过管道将数据传输到 Estimator 中：

```
def train_input_fn(features, labels, batch_size):
    """An input function for training"""
    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))

    # Shuffle, repeat, and batch the examples.
    dataset = dataset.shuffle(1000).repeat().batch(batch_size)

    # Build the Iterator, and return the read end of the pipeline.
    return dataset.make_one_shot_iterator().get_next()
```

- 三个参数：
  - features：包含原始输入特征的 {'feature_name':array} 字典（或 DataFrame）。
  - labels：包含每个样本的标签的数组。
  - batch_size：表示所需批次大小的整数。

- 切片
  - 在最简单的情况下，tf.data.Dataset.from_tensor_slices 函数接受一个数组并返回表示该数组切片的 tf.data.Dataset。
  - 例如，一个包含 mnist 训练数据的数组的形状为 (60000, 28, 28)。将该数组传递给 from_tensor_slices，会返回一个包含 60000 个切片的 Dataset 对象，其中每个切片都是一个 28x28 的图像。

```
train, test = tf.keras.datasets.mnist.load_data()
mnist_x, mnist_y = train

mnist_ds = tf.data.Dataset.from_tensor_slices(mnist_x)
print(mnist_ds)
```

  - 返回结果：<TensorSliceDataset shapes: (28, 28), types: tf.uint8>
  - 显示数据集中条目的形状和类型
  - 请注意，数据集不知道自己包含多少条目

  - 确保 features 是标准字典，然后您就可以将数组字典转换为字典 Dataset

```
features = {("SepalLength", np.array(  1.4, 2.0 )),
            ("PetalWidth", np.array(  3.0, 4.0 )),
            ("PetalLength", np.array(  5.0, 6.0 )),
            ("SepalWidth", np.array(  7.0, 8.0 ))}
dataset = tf.data.Dataset.from_tensor_slices(dict(features))
print(dataset)
```

  - 返回结果：

```
<TensorSliceDataset
    shapes: {
        SepalLength: (), PetalWidth: (),
        PetalLength: (), SepalWidth: ()},
    types: {
        SepalLength: tf.float64, PetalWidth: tf.float64,
        PetalLength: tf.float64, SepalWidth: tf.float64}
>
```

  - 如果 Dataset 包含结构化元素，则 Dataset 的 shapes 和 types 将采用同一结构
  - 此数据集包含所有类型为 tf.float64 的标量的字典

  - train_input_fn 的第一行使用相同的功能，但添加了另一层结构
  - 它会创建一个包含 (features, labels) 对的数据集

```
# Convert the inputs to a Dataset.
dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))
print(dataset)
```

  - 返回结果：

```
<TensorSliceDataset
  shapes: (
    {
      SepalLength: (), PetalWidth: (),
      PetalLength: (), SepalWidth: ()},
    ()),

  types: (
    {
      SepalLength: tf.float64, PetalWidth: tf.float64,
      PetalLength: tf.float64, SepalWidth: tf.float64},
    tf.int64)>
```

  - 其中，标签是类型为 int64 的标量

- 操作
  - Dataset 会按固定顺序迭代数据一次，并且一次仅生成一个元素
  - 它需要进一步处理才可用于训练
  - tf.data.Dataset 类提供了方法来更好地准备用于训练的数据

```
# Shuffle, repeat, and batch the examples.
dataset = dataset.shuffle(1000).repeat().batch(batch_size)
```

- **shuffle** 方法使用一个固定大小的缓冲区，在条目经过时执行随机化处理
  - 将 buffer_size 设置为大于 Dataset 中样本数的值，可确保数据完全被随机化处理
- **repeat** 方法会在结束时重启 Dataset，要限制周期数量，请设置 count 参数。
- **batch** 方法会收集大量样本并将它们堆叠起来以创建批次。
  - 这为批次的形状增加了一个维度。新的维度将添加为第一个维度。

  - 以下代码对之前的 MNIST Dataset 使用 batch 方法。这样会产生一个包含表示 (28,28) 图像堆叠的三维数组的 Dataset：

```
print(mnist_ds.batch(100))
```

  - 返回结果：<BatchDataset shapes: (?, 28, 28), types: tf.uint8>
  - 请注意，该数据集的批次大小是未知的，因为最后一个批次具有的元素数量会减少。

  - 在 train_input_fn 中，经过**批处理**之后，Dataset 包含元素的一维向量，其中每个标量之前如下所示

```
<TensorSliceDataset
  shapes: (
    {
      SepalLength: (?,), PetalWidth: (?,),
      PetalLength: (?,), SepalWidth: (?,)},
    (?,)),

  types: (
    {
      SepalLength: tf.float64, PetalWidth: tf.float64,
      PetalLength: tf.float64, SepalWidth: tf.float64},
    tf.int64)>
```

- 返回
  - 每个 Estimator 的 train、evaluate 和 predict 方法都需要输入函数返回包含 TensorFlow 张量的 (features, label) 对
  - train_input_fn 使用以下行将数据集转换为所需格式：

```
# Build the Iterator, and return the read end of the pipeline.
features_result, labels_result = dataset.make_one_shot_iterator().get_next()
```

  - 结果会生成与 Dataset 中条目的布局相匹配的 TensorFlow 张量结构。要简要了解这些对象及其使用方式，请参阅简介。

```
print((features_result, labels_result))
```

  - 结果：

```
({
  'SepalLength': <tf.Tensor 'IteratorGetNext:2' shape=(?,) dtype=float64>,
  'PetalWidth': <tf.Tensor 'IteratorGetNext:1' shape=(?,) dtype=float64>,
  'PetalLength': <tf.Tensor 'IteratorGetNext:0' shape=(?,) dtype=float64>,
  'SepalWidth': <tf.Tensor 'IteratorGetNext:3' shape=(?,) dtype=float64>},
 Tensor("IteratorGetNext_1:4", shape=(?,), dtype=int64))
```

2、读取CSV文件

构建从本地CSV文件读取数据且兼容 Estimator 的输入函数

- 构建Dataset：
  - 构建一个 TextLineDataset 对象来实现一次读取文件中的一行数据
  - 通过调用 skip 方法来跳过文件的第一行标题

```
ds = tf.data.TextLineDataset(train_path).skip(1)
```

- 构建 csv 行解析器，解析数据集中的每一行

```
# Metadata describing the text columns
COLUMNS = ['SepalLength', 'SepalWidth',
           'PetalLength', 'PetalWidth',
           'label']
FIELD_DEFAULTS = [[0.0], [0.0], [0.0], [0.0], [0]]
def _parse_line(line):
    # Decode the line into its fields
    fields = tf.decode_csv(line, FIELD_DEFAULTS)

    # Pack the result into a dictionary
    features = dict(zip(COLUMNS,fields))

    # Separate the label from the features
    label = features.pop('label')

    return features, label
```

- 解析行
  - 数据集提供很多用于在通过管道将数据传送到模型的过程中处理数据的方法
  - 最常用的方法是 map，它会对 Dataset 的每个元素应用转换
  - map 方法接受 map_func 参数，此参数描述如何转换 Dataset 中的每个条目

  - 从 csv 文件中流式传出行时对行进行解析。将 _parse_line 函数传递给 map 方法：

```
ds = ds.map(_parse_line)
print(ds)
```

  - 结果为：

```
<MapDataset
shapes: (
  {SepalLength: (), PetalWidth: (), ...},
  ()),
types: (
  {SepalLength: tf.float32, PetalWidth: tf.float32, ...},
  tf.int32)>
```

  - 现在，数据集包含 (features, label) 对，而不是简单的标量字符串

- 试试看：替换train_input_fn

```
train_path, test_path = iris_data.maybe_download()

# All the inputs are numeric
feature_columns = [
    tf.feature_column.numeric_column(name)
    for name in iris_data.CSV_COLUMN_NAMES[:-1]]

# Build the estimator
est = tf.estimator.LinearClassifier(feature_columns,
                                    n_classes=3)

# Train the estimator
batch_size = 100
est.train(
    steps=1000,
    input_fn=lambda : iris_data.csv_input_fn(train_path, batch_size))
```

- Estimator 要求 input_fn 不接受任何参数。为了不受此限制的束，我们使用 lambda 来获取参数并提供所需的接口。

3、总结

- tf.data 模块提供一系列类和函数，可用于轻松从各种来源读取数据
- tf.data 还提供简单而又强大的方法，用于应用各种标准和自定义转换
- 更多学习：
  - Creating Custom Estimators：如何自行构建自定义 Estimator 模型
  - Low Level Introduction：如何使用 TensorFlow 的低阶 API 直接尝试 tf.data.Datasets
  - Importing Data：介绍Datasets的其他功能

更新时间：2018年04月23日14:01:16