



黑马程序员
www.itheima.com

传智播客旗下
高端IT教育品牌

Tensorflow 与深度学习



课程第五天 - 线程队列与 IO 操作

1、队列和线程

2、文件读取

3、图片处理

在计算争分夺秒的时候，需要去提高 IO 读取的速度？

队列和线程

- 1、队列与队列管理器
- 2、线程和协调器

Tensorflow 队列

- 在训练样本的时候，希望读入的训练样本时有
序的
- `tf.FIFOQueue` 先进先出队列，按顺序出队列
- `tf.RandomShuffleQueue` 随机出队列

tf.FIFOQueue

- `FIFOQueue(capacity, dtypes, name='fifo_queue')`
- 创建一个以先进先出的顺序对元素进行排队的队列
 - `capacity` : 整数。可能存储在此队列中的元素数量的上限
 - `dtypes` : `DType` 对象列表。长度 `dtypes` 必须等于每个队列元素中的张量数, `dtype` 的类型形状, 决定了后面进队列元素形状
- `method`
- `dequeue(name=None)`
- `enqueue(vals, name=None):`
- `enqueue_many(vals, name=None):vals` 列表或者元组
返回一个进队列操作
- `size(name=None)`

- 完成一个出队列、+1、入队列操作（同步操作）

入队列需要注意

```
import tensorflow as tf

# 指定类型
Q = tf.FIFOQueue(5, dtypes=tf.float32)
# [, ] 用逗号隔开指定是列表，防止与 tensor 混淆
init = Q.enqueue_many([[0.1, 0.2, 0.3], ])
```


分析：当数据量很大时，入队操作从硬盘中读取数据，放入内存中，主线程需要等待入队操作完成，才能进行训练。会话里可以运行多个线程，实现异步读取。

队列管理器

- `tf.train.QueueRunner(queue, enqueue_ops=None)`
创建一个 QueueRunner
 - `queue` : A Queue
 - `enqueue_ops` : 添加线程的队列操作列表, `[]*2`, 指定两个线程
 - `create_threads(sess, coord=None, start=False)`
创建线程来运行给定会话的入队操作
 - `start` : 布尔值, 如果 True 启动线程; 如果为 False 调用者必须调用 `start()` 启动线程
 - `coord`: 线程协调器, 后面线程管理需要用到
 - `return` :

通过队列管理器来实现变量加 1，入队，主线程出队列的操作，观察效果？
(异步操作)

分析：这时候有一个问题就是，入队自顾自的去执行，在需要的出队操作完成之后，程序没法结束。需要一个实现线程间的同步，终止其他线程。

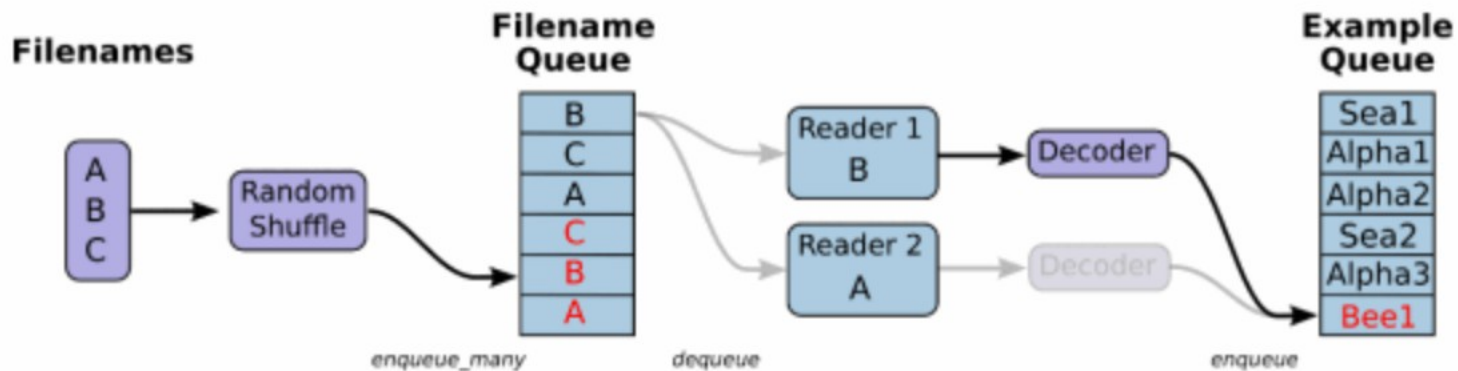
线程协调器

- `tf.train.Coordinator()`
线程协调员，实现一个简单的机制来协调一组线程的终止
 - `request_stop()`
 - `should_stop()` 检查是否要求停止
 - `join(threads=None, stop_grace_period_secs=120)`
等待线程终止
- return: 线程协调员实例

文件读取

- 1、文件读取流程
- 2、文件读取 API
- 3、文件读取案例

文件读取流程



1、文件读取 API- 文件队列构造

- `tf.train.string_input_producer(string_tensor, shuffle=True)`

将输出字符串（例如文件名）输入到管道队列

- `string_tensor` 含有文件名的 1 阶张量
- `num_epochs`: 过几遍数据，默认无限过数据
- `return`: 具有输出字符串的队列

2、文件读取 API- 文件阅读器

- 根据文件格式，选择对应的文件阅读器
- `tf.TextLineReader`
 - 阅读文本文件逗号分隔值（CSV）格式，默认按行读取
 - `return`：读取器实例
- `tf.FixedLengthRecordReader(record_bytes)`
 - 要读取每个记录是固定数量字节的二进制文件
 - `record_bytes`: 整型，指定每次读取的字节数
 - `return`：读取器实例
- `tf.TFRecordReader`
 - 读取 `TfRecords` 文件
- 有一个共同的读取方法：
- `read(file_queue)`：从队列中指定数量内容
返回一个 `Tensors` 元组（`key` 文件名字，`value` 默认的内容（行，字节））

3、文件读取 API- 文件内容解码器

- 由于从文件中读取的是字符串，需要函数去解析这些字符串到张量
- `tf.decode_csv(records, record_defaults=None, field_delim=None, name=None)`
将 CSV 转换为张量，与 `tf.TextLineReader` 搭配使用
 - `records`: tensor 型字符串，每个字符串是 csv 中的记录行
 - `field_delim`: 默认分割符“,”
 - `record_defaults`: 参数决定了所得张量的类型，并设置一个值在输入字符串中缺少使用默认值，如
- `tf.decode_raw(bytes, out_type, little_endian=None, name=None)`
将字节转换为一个数字向量表示，字节为一字符串类型的张量，与函数 `tf.FixedLengthRecordReader` 搭配使用，二进制读取为 `uint8` 格式

开启线程操作

- `tf.train.start_queue_runners(sess=None, coord=None)`
收集所有图中的队列线程，并启动线程
 - `sess`: 所在的会话中
 - `coord`: 线程协调器
 - `return`: 返回所有线程队列

如果读取的文件为多个或者样本数量为多个，怎么去管道读取？

管道读端批处理

- `tf.train.batch(tensors, batch_size, num_threads = 1, capacity = 32, name=None)`
 - 读取指定大小（个数）的张量
 - `tensors`：可以是包含张量的列表
 - `batch_size`：从队列中读取的批处理大小
 - `num_threads`：进入队列的线程数
 - `capacity`：整数，队列中元素的最大数量
 - `return:tensors`
- `tf.train.shuffle_batch(tensors, batch_size, capacity, min_after_dequeue, num_threads=1,)`
 - 乱序读取指定大小（个数）的张量
 - `min_after_dequeue`：留下队列里的张量个数，能够保持随机打乱

文件读取案例

- 1、文件简单读取
- 2、CIFAR-10 二进制数据读取

文件读取案例流程

CIFAR-10 二进制数据读取

图像读取

- 1、图像基本知识
- 2、图像读取 API
- 3、TFRecords 分析、存取

图像基本知识



图像数字化三要素

- 三要素：长度、宽度、通道数

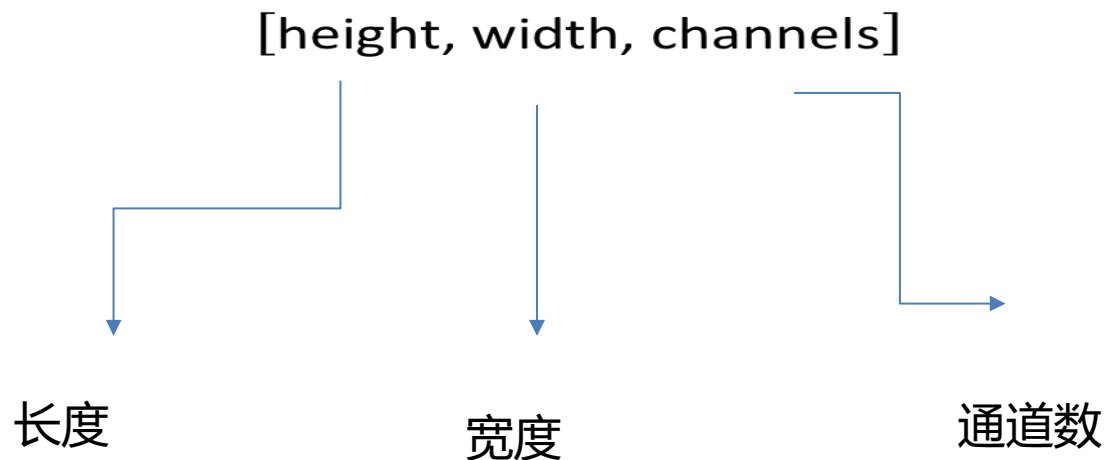


一通道，灰度值

三通道，RGB

三要素与张量的关系

指定 3-D 张量：



图像基本操作

目的：

- 1、增加图片数据的统一性
- 2、所有图片转换成指定大小
- 3、缩小图片数据量，防止增加开销

操作：

- 1、缩小图片大小

图像基本操作 API

- `tf.image.resize_images(images, size)`
缩小图片
 - `images` : 4-D 形状 [batch, height, width, channels] 或 3-D 形状的张量 [height, width, channels] 的图片数据
 - `size` : 1-D int32 张量: `new_height, new_width` , 图像的新尺寸
 - 返回 4-D 格式或者 3-D 格式图片

图片批处理案例

狗图片读取

图像读取 API

- 图像读取器
- `tf.WholeFileReader`
 - 将文件的全部内容作为值输出的读取器
 - `return` : 读取器实例
 - `read(file_queue)`: 输出将是一个文件名（`key`）和该文件的内容（值）
- 图像解码器
- `tf.image.decode_jpeg(contents)`
 - 将 JPEG 编码的图像解码为 `uint8` 张量
 - `return`: `uint8` 张量， 3-D 形状 [`height`, `width`, `channels`]
- `tf.image.decode_png(contents)`
 - 将 PNG 编码的图像解码为 `uint8` 或 `uint16` 张量
 - `return`: 张量类型， 3-D 形状 [`height`, `width`, `channels`]

图片批处理案例流程

- 1、构造图片文件队列
- 2、构造图片阅读器
- 3、读取图片数据
- 4、处理图片数据

TFRecords 分析、存取

- TFRecords 是 Tensorflow 设计的一种内置文件格式，是一种二进制文件，它能更好的利用内存，更方便复制和移动
- 为了将二进制数据和标签 (训练的类别标签) 数据存储储在同一个文件中

TFRecords 文件分析

- 文件格式: *.tfrecords
- 写入文件内容: Example 协议块

TFRecords 存储

1、建立 TFRecord 存储器

- `tf.python_io.TFRecordWriter(path)`
写入 tfrecords 文件
 - path: TFRecords 文件的路径
 - return : 写文件
- method
 - `write(record)`: 向文件中写入一个字符串记录
 - `close()`: 关闭文件写入器

注：字符串为一个序列化的 `Example`, `Example.SerializeToString()`

TFRecords 存储

2、构造每个样本的 Example 协议块

- `tf.train.Example(features=None)`
 - 写入 tfrecords 文件
 - `features:tf.train.Features` 类型的特征实例
 - `return` : `example` 格式协议块
- `tf.train.Features(feature=None)`
 - 构建每个样本的信息键值对
 - `feature`: 字典数据, `key` 为要保存的名字, `value` 为 `tf.train.Feature` 实例
 - `return:Features` 类型
- `tf.train.Feature(**options)`
 - `**options` : 例如
`bytes_list=tf.train. BytesList(value=[Bytes])`
`int64_list=tf.train. Int64List(value=[Value])`
- `tf.train. Int64List(value=[Value])`
- `tf.train. BytesList(value=[Bytes])`
- `tf.train. FloatList(value=[value])`

TFRecords 读取方法

- 同文件阅读器流程，中间需要解析过程
- 解析 TFRecords 的 example 协议内存块
- `tf.parse_single_example(serialized, features=None, name=None)`
 - 解析一个单一的 Example 原型
 - `serialized`：标量字符串 Tensor，一个序列化的 Example
 - `features`：dict 字典数据，键为读取的名字，值为 FixedLenFeature
 - `return`：一个键值对组成的字典，键为读取的名字
- `tf.FixedLenFeature(shape, dtype)`
 - **shape**：输入数据的形状，一般不指定，为空列表
 - **dtype**：输入数据类型，与存储进文件的类型要一致
类型只能是 float32, int64, string

CIFAR-10 批处理结果存入 tfrecords 流程

- 1、构造存储器
- 2、构造每一个样本的 Example
- 3、写入序列化的 Example

读取 tfrecords 流程

- 1、构造 TFRecords 阅读器
- 2、解析 Example
- 3、转换格式， bytes 解码



Thank You!

改变中国 IT 教育，我们正在行动

www.itcast.cn