Tensorflow 分布式训练

1，PS-worker 架构

将模型维护和训练计算解耦合，将模型训练分为两个作业（job）：

- 模型相关作业，模型参数存储、分发、汇总、更新，有由 PS 执行
- 训练相关作业，包含推理计算、梯度计算（正向/反向传播），由 worker 执行

该架构下，所有的 woker 共享 PS 上的参数，并按照相同的数据流图传播不同 batch 的数据，计算出不同的梯度，交由 PS 汇总、更新新的模型参数，大体逻辑如下：

1. pull：各个 woker 根据数据流图拓扑结构从 PS 获取最新的模型参数
2. feed：各个 worker 根据定义的规则填充各自 batch 的数据
3. compute：各个 worker 使用第一步的模型参数计算各自的 batch 数据，求出各自 batch 的梯度
4. push：各个 worker 将各自的梯度推送到 PS
5. update：PS 汇总来自 n 个 worker 的 n 份梯度，求出平均值后更新模型参数

分布式经典架构 PS-worker 会重复上面步骤，直到损失到达阈值或者轮数到达阈值。

2，数据并行模式分类

根据数据流图构建模式分类：

- 图内复制：单进程、‘单机多卡’的数据并行训练，需要用户自己实现梯度汇总和均值计算。实例，models/tutorials/image/cifar10/cifer10_multi_gpu-train.py（见下节）
- 图间复制：多进程、跨多机的分布式训练，使用同步优化器（SyncReplicasOptimizer）实现分布式梯度计算和模型参数更新。实例，tensorflow/tools/dist_test/python/mnist_replica.py（分布式同步训练实践，见下下节）

根据参数更新机制分类：

- 异步训练：各个 worker 独立训练，计算出梯度后即刻更新参数，不需要等待其他 worker 完成计算
- 同步训练：所有 worker 完成本轮计算后，汇总梯度，更新模型，计算能力强的 worker 需要阻塞等待其他 worker

两种训练机制同时支持上面两周数据流图构建模式。一般来说同步机制收敛快，异步单步计算快，但易受单批数据影响，不稳定。

3，同步优化器

tensorflow 进行**同步**（同步训练模式专用）各个 worker 梯度并进行优化时，会使用特殊的优化器即同步优化器，tf.train.SyncReplicasOptimizer，其**第一个参数为普通优化器**，我们可以定义一个普通的优化器传入，后续参数如下：

参数名称	功能说明	默认值
replicas_to_aggragate	并行副本数	num_workers
total_num_replicas	实际副本数（worker 数目）	num_workers

并行副本数指期望的每一步中并行的 batch 数据数目，实际副本数指参与的 workers 数目，

- 并行=实际：全民参与，一个 worker 领取一个 batch 数据
- 并行>实际：能者多劳，先完成自己 batch 的 worker 会继续领取未训练数据，PS 会等到梯度份数到达并行数后进行模型参数计算
- 并行<实际：替补等位，存在空闲的 worker，取代可能出现的异常 worker，确保训练过程高可用

运算过程

- 计算梯度过程同普通优化器，调用基类的 Optimizer 的 compute_gradients 成员方法
- 更新参数时重写了 Optimizer 的 apply_gradients 方法，见 tensorflow/python/training/sync_replicas_optimizer.py

讲解同步优化器工作逻辑之前，介绍两个概念，

梯度聚合器

每一个模型参数有一个自己队列，收集来自不同 worker 的梯度值，梯度聚合器包含 M 个队列对应 M 个模型参数，每个队列收集来自 N 个 worker 计算出来的 N 个梯度值。

同步标记队列

存储同步标记，实际上就是 N 个 global_step 值，每个 worker 领取一个，用于控制同步

以全民参与模式为例

worker 工作模式如下：

1. 从同步标记队列领取一个 global_step，表示全局训练步数的同步标记
2. 将同步标记值赋予 worker 的本地训练步数 local_step

3. 从 PS 获取最新模型参数
4. 计算出 M 个梯度值
5. 将 M 个梯度值推送到 PS 上的 M 个梯度队列中

PS 工作模式如下：

1. 从梯度聚合器上收集 worker 推送过来的梯度值，每个队列收集 N 份（对应 N 个 global_step 下训练值）后，计算均值，收集齐 M 个均值后，得到 M 对{模型参数，梯度值}的聚合元组
2. 更新模型参数
3. 向同步标记队列推送 N 个 global_step+1 标记

聚合器收集梯度值并校验 local_step 是否符合 global_step，是则接收梯度值，计算能力强的 worker 提交梯度后由于没有同步标记可以领取所以被阻塞，PS 集齐 N 份后更新参数，发布下次 N 个同步标记，开始下一步训练。

由于初始 PS 不会更新参数发布同步标记，所以需要初始化同步标记队列——sync_init_op，直接向队列注入 N 个 0 标记。

分布式模型训练需要的主要初始化操作如下（opt.tf.train.SyncReplicasOptimizer）：

操作名称	常用变量名	功能说明
opt.local_step_init_op	local_init_op	loacl_step 初始值
pot.chief_init_op	local_init_op	gobal_step 初始值
opt.ready_for_local_init_op	ready_for_local_init_op	为未初始化的 Variable 设置初始值
opt.get_chief_queue_runner	chief_queue_runner	同步标记队列启动 QueueRunner 实例
opt.get_init_tockens_op	sync_init_op	同步标记队列初始化
tf.global_variables_initializer	init_op	全局 Variable 设置初始值

如果使用模型管理类 Supervsor，可以将大部分工作交由其代劳。

以能者多劳模式对比

模型参数个数 M，worker 个数 N，并行副本数 R（R>N），此时

梯度聚合器仍然有 M 个参数收集队列，每一个队列要收集 R 份才进行汇总，R>N 所以会存在某个 worker 领取多份数据的情况。

同步标记队列存储 R 个同步标记，以确保每一步中梯度聚合器可以收集到 R 份数据。

4，异步优化器

异步优化器没有很多附加参量，和单机训练几乎一致，只是每个 worker 获取参数需要从另一个进程 PS 中得到而已。

5，模型管理类 Supervsor

本质上是对 Saver（模型参数存储恢复）、Coordinator（多线程服务生命周期管理）、SessionManager（单机以及分布式会话管理）三个类的封装，Coordinator 会监测程序的线程是否运行正常，任何异常的出现都会向 Supervisor 报告，此时 Coordinator 讲程序的停止条件设置为 True，Supervisor 停止训练并清理工作（关闭会话、回收内存等），其他服务检测到 True 后会各自关闭服务，终止线程。

SessionManager 帮助用户创建管理单机或是分布式会话，以便简化数据流图的生命周期和维护逻辑，同事负责将 checkpoint 文件中加载出的数据恢复到数据流图中。

流程逻辑如下：

1. 创建 Supervisor 实例，构造方法需要传入 checkpoint 文件和 summary 文件存储目录（Supervisor 的 logdir 参数）
2. 调用 tf.train.Supervisor.managed_session，从 Supervisor 实例获取会话实例
3. 使用该会话执行训练，训练中需要检查停止条件，保证训练正确性

获取 managed_session 时，Supervisor 会通过 QueueRunner 同时启动一下三个服务：

- 检查点服务：将数据流图中的参数定期保存，默认 10min 保存一次，且会识别 global_step（Supervisor 的 global_step 参数）
- 汇总服务：默认 2min 一次
- 步数计数器服务：向汇总添加 global_step/sec，2min 一次

使用 managed_session 创建会话时，会自动恢复上一次的结果并继续训练

一、tensorflow GPU 设置

GPU 指定占用

```
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.7)
sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
```

上面分配给 tensorflow 的 GPU 显存大小为：GPU 实际显存*0.7。

GPU 模式禁用

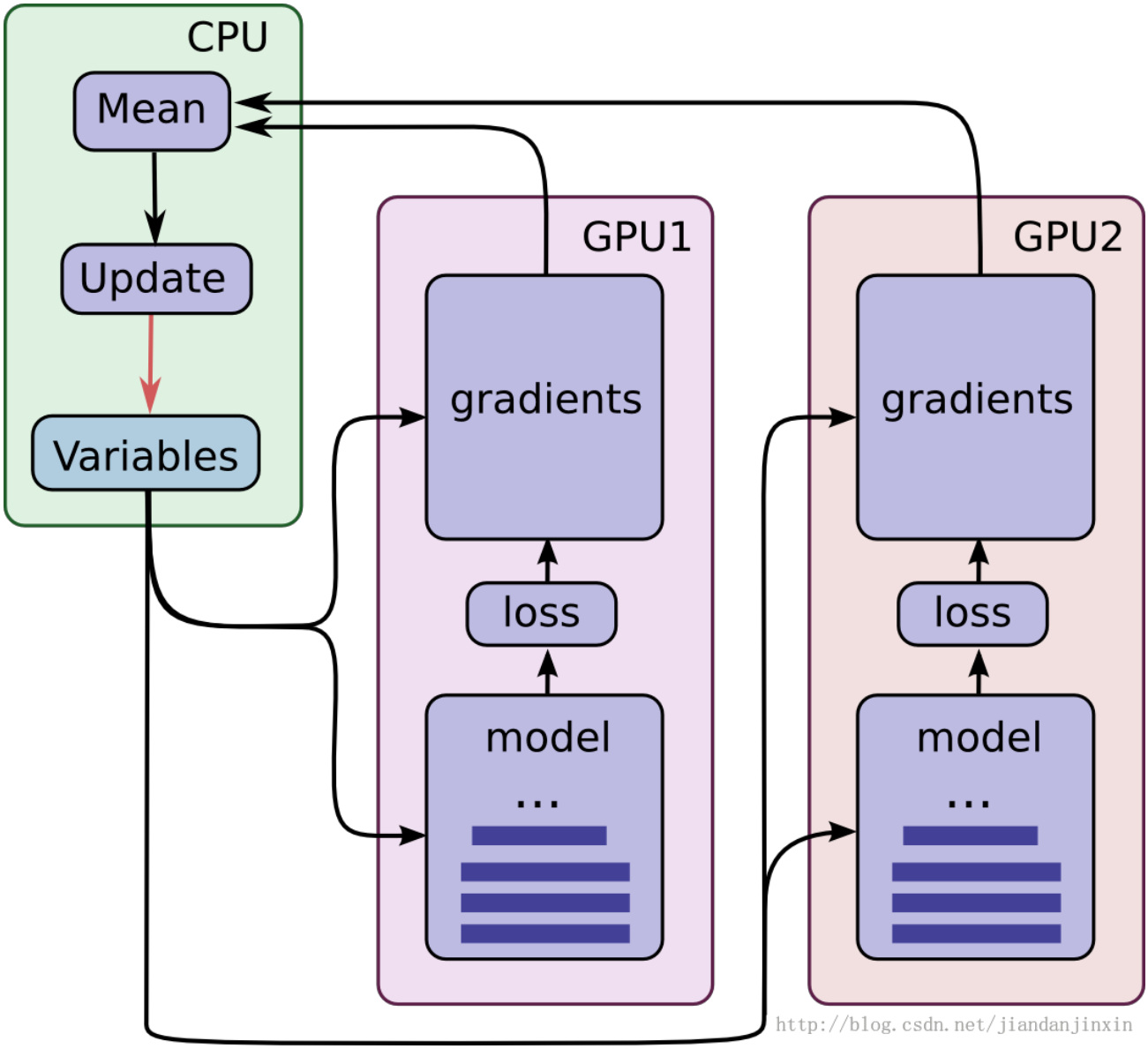
```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="-1"
```

GPU 资源申请规则

```
# 设置 GPU 按需增长
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
```

二、单机多 GPU 工作原理

以一篇 csdn 博客（出处见水印）上的图说明多 GPU 工作原理：



想让 TensorFlow 在多个 GPU 上运行, 需要建立 multi-tower 结构, 在这个结构里每个 tower 分别被指配给不同的 GPU 运行，汇总工作一般交由 CPU 完成，示意如下，

```
# 新建一个 graph.
c = []
for d in ['/gpu:2', '/gpu:3']:
  with tf.device(d):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
    c.append(tf.matmul(a, b))
with tf.device('/cpu:0'):
  sum = tf.add_n(c)
# 新建 session with log_device_placement 并设置为 True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# 运行这个 op.
print sess.run(sum)
```

13	