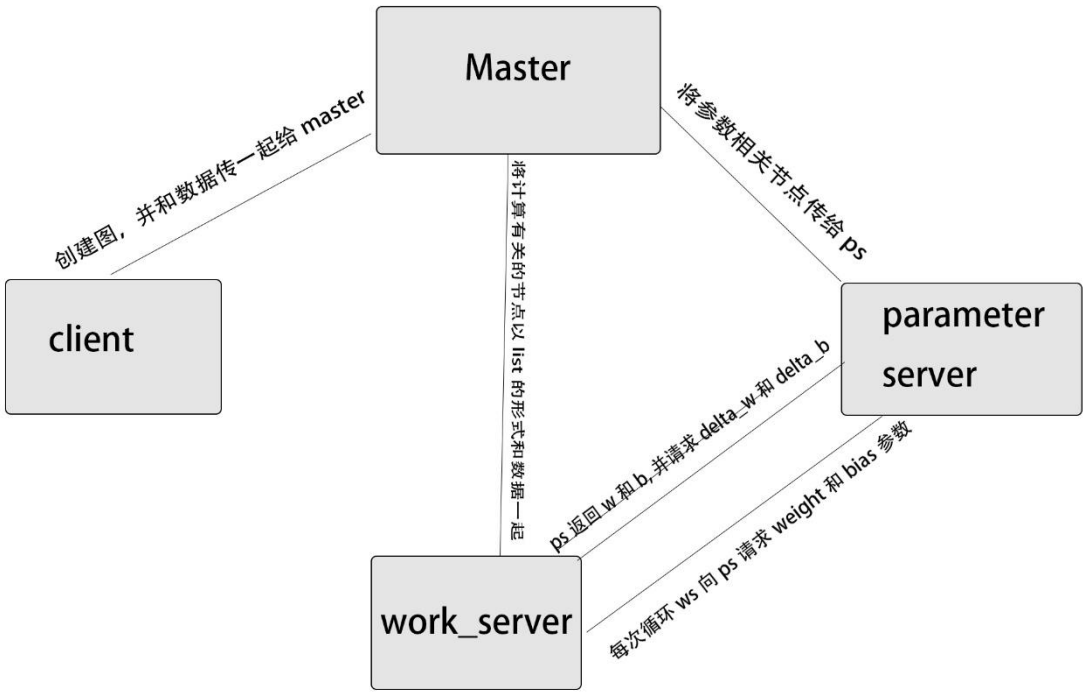
模拟 TensorFlow 分布式实验文档

16302010033 孔庆灿

一、整体设计架构



我的设计其实和文档上要求的那种结构比较相似，不同更多的体现在细节的实现上。

首先，**client 端**要做的是决定整个训练网络的结构，包括输入层个数、隐藏层个数、输出层个数和学习率。然后根据用户定义的网络结构创建 client 类的对象。Client 类里面有个方法可以将用户定义的网络结构转换成图的形式。同时 client 端有方法可以读取 CSV 数据，并转化成训练需要的数据格式。然后 client 端通过与 master 端建立连接，将 client 对象生成的图和需要训练的数据传给 master 端，然后等待 master 返回最终的训练结果(准确率)Ps:我整个图的构造时采用二叉树的形式，每个节点储存自己的 name、value 以及自己的子节点还有计算自己的方法名称。

然后呢，在 **master 端**，连接 client 端，接受 client 发送过来的图和训练数据。然后 master 创建与 worker_server 和 parameter_server 的连接。然后 master 遍历整个图，将运算相关的节点以 list 的形式和训练数据一起传给 worker_server，将参数有关的节点（weight 和 bias）传给 parameter_server。Master 的主要工作其实就是分配任务，让 worker_server 计算，让 parameter_server 给 worker_server 传数据。做完这些工作之后，master 进程就等待 worker 返回的训练结果就可以了，最后在将结果返回给 client。

在 **worker_server 端**，首先要做的工作是建立好和 master、parameter_server 之间的连接，然后接收 master 端传送过来的运算节点的 list 以及训练数据。在 worker_server 端定义了需要的一些运算的方法，然后将这些方法以键值对的形式和名字字符串绑定然后储存在 python 字典里。然后向 parameter_server 请求需要用到的 weight 和 bias 参

数，并将这些参数赋给相应的节点。之后以从二叉树最下面的运算节点开始遍历的方式，通过节点里储存的运算方法名字，根据之前定义的函数字典来解析出用到的函数，然后从最底部向上递归地算出最终的运算结果。最后通过计算残差来计算需要更新的 weight 的大小，即 delta_weight。每一次运算后将 delta_weight 和 delta_bias 传给 parameter_server，让他更新相应的参数。

最后，在 **parameter_server 端，**它主要负责的是通过和 master 端连接获得参数的一些初始值，然后将这些参数储存在自己的变量里。然后等待 worker_server 端发出请求参数的请求，将保存在 parameter_server 中的 weight 和 bias 发送到 worker-server 中，并且等待接受 worker_server 端发送过来的 delta_weight、delta_bias 来更新自己储存的值。

二、代码结构

Client 端	
类 Client	里面主要的方法是 generate_graph（），可以通过用户设置的网络结构来生成一个二叉树来代表图
Read_file()	用来读取外部文件的数据并储存在一个列表中返回出来
Deal_inputs()	处理读入的数据，将数据以（输入值，目标结果值）的形式分开，并储存返回

Worker 端	
Sigmoid()	神经元的激活函数，前向传播时要用
Matmul()	矩阵相乘的函数，用于计算中间节点的值
Plus()	矩阵相加函数
Loss_function_der()	损失函数的求导，用来计算残差
Sigmoid_der()	激活函数求导，用来计算残差
Calculate_output()	解析计算节点所需要的计算方法，并逐级算出各个节点的值

Session.py	
Register()	注册端口号，创建任务队列，并将创建的任务队列返回
Connect()	根据地址和端口号与另一个客户端连接，并得到相应的任务队列来传送数据

对于 master 类和 parameter 类，只是创建了一个 main 方法里面用来和其他服务器连接以及相应的传送数据。

三、程序解释

我的程序的结构和运行逻辑在第一部分已经说得比较具体了，所以在这一部分就不赘述了。至于为什么这么设计，主要有两个原因吧。首先是通过读 lab 说明文当中的要求对整个项目的大体结构有了比较具体的印象，得到大概是需要 client、master、worker_server、parameter_server 这四个主要端系统。因为单单是读文档并没有对 TensorFlow 的分布式并

没有比较好的理解，所以又到网上找了 TensorFlow 分布式原理的一些资料。现在我对 TensorFlow 的分布式的理解是：因为 TensorFlow 本来就是用的图运算，用户定义的运算结构都是以图的形式在 TensorFlow 中储存着的，变量在图中相当于占位符。对于图中每个节点的运算，只需要把计算该节点的参数全都 feed 好，通过 session.run()就能计算出该节点的值。所以对于 TensorFlow 来说，不管是整个运算图还是子图，只要占位符都设置好值之后，都可以运算出结果。这也是为什么 master 能进行分配任务的原因。其实在 TensorFlow 里，session 一定程度上也就相当于 master 的作用来给不同机器分配运算子图来分配任务。在 TensorFlow 的实现中，参数是分块地保存在 parameter_server 服务器中的，一个计算任务可以通过拆分数据来分给不同的 worker_server，每个 worker_server 所需要的参数对应于 parameter_server 中的一个数据块，每个 worker_server 执行计算任务，需要更新 weight 的时候，就由 worker_server 给 parameter_server 发送 delta_weght 和 delta_bias 来对参数进行更新。我的设计中就是模拟了上面的参数更新和参数传递的过程。

在设计的细节中，因为我们很难向 TensorFlow 一样根据用户设置的各种运算来自动生成一个符合条件的图，所以只能是对网络的层数进行一定程度的限制，来让用户定义一个这种符合限制的图，这样做的坏处就是可扩展性不太强，但是虽然不能更改网络层数，神经元个数却可以由用户随意设置，也算是有一定的灵活性。其次呢，因为参数更新时一次性涉及很多节点的输入，仅仅通过一个自动生成的二叉树很难获得需要的每个节点的准确的值，并且一系列的求导运算也是相当复杂，所以对于反向传播更改 weight 时并没有根据图来计算，而是通过一些串行的运算来实现的。这样虽然没有完全模拟出 TensorFlow 图算法的全部，但在前向传播时也的确和图算法基本类似，也算完成了图分割，任务分配的作用。所以说，我的程序虽然对 TensorFlow 的整体运行机制没有完全模拟出来，但是它分布式的基本思想已经能比较好的体现了，也能通过我的程序比较好地理解分布式。

四、遇到的问题和解决方法

1、 不知道 TensorFlow 的分布式是怎么样的，不知道从什么地方开始入手
解决方法：仔细阅读了几遍需求文档，并且在网上搜了很多关于 TensorFlow 图算法和分布式原理的文章
2、 一个系统不知道怎么模拟分布式
解决方法：刚开始想的是通过面向对象的方式，一个对象代表一个端系统，通过对象间函数传值的方式来传递数据。后来实现了之后，想了想这样给我多个机器并不能很好地分配任务。后来在网上找到了用多进程模拟分布式的方法。
3、 不知道以什么样的形式构建图
解决方法：刚开始没有思路，就去看了 TensorFlow 的 graph，后来感觉像他们那种图我根本实现不了，后来受同学启发，用二叉树的形式构建图。
4、 传递过去一个图之后不知道怎么去运算
解决方法：从网上知道 python 可以传函数作为参数，并且函数可以保存到字典里，所以对于需要计算的节点只需要把该节点的计算方法的名字传进去，到了需要计算的时候再根据函数名、函数字典解析出函数来进行计算
5、 不清楚 parameter_server 的具体任务，感觉它可有可无
解决方法：通过查阅更多的 TensorFlow 分布式原理的资料，得到他的任务就是通过接收 worker 传过来的 delta_weight 来进行参数的简单更新。
6、 关于怎么通过图来进行反向更新权值十分困惑
解决方法：图只用来正向传播计算，反向传播采用串行的计算

7、 两个服务器间的连接和数据通信出现问题

解决方法：仔细看了相关教程，大致了解了其内部机制，之后连接和通信就比较简单了