Project1 实验文档

目录

一、 系统介绍与使用说明

1. 整体介绍

该系统为一个简单的分布式机器学习系统，支持以分布式的模式将机器学习的过程部署到多台机器上执行。当机器学习需训练的参数量大，计算过程多时，可通过配置集群将参数集中在一台机器上进行更新，并配置多台 worker 同时进行计算以提高训练速度。

该系统以 python 包的形式提供，将该包导入工程目录并在 python 文件中引入后，client 端可动态配置集群以初始化 cient，服务端通过调用注册服务的方式配置为分配服务器、参数服务器或计算服务器，并运行以提供服务。client 定义初始化参数和训练方法后，迭代运行，即可开始训练。

该系统目前只支持了以简单的反向传播算法进行训练，拓展后可支持更多类型的机器学习方法。

2. 使用说明

为测试该系统，提供了测试案例——1. 利用该系统对 student_data.csv 文件进行数据读取训练，通过 GRE Scores (gre), GPA Scores (gpa), Class rank (rank) 数据预测能否被录取。 2. 利用此系统对鸢尾花数据集进行分类。

运行步骤：

- 运行环境为 python3 环境，需安装包 numpy，google，grpcio
- 运行 master.py、parameter_server.py、worker.py、worker1.py（三者运行顺序无要求，但需保证 classify.py 运行前三者均处于正常运行状态）
- 运行 classify.py（student_data 分类）或 iris_classify.py(鸢尾花分类)
- （由于项目的 python 虚拟环境太大所以没有提交…,如果无法运行的话我可以再把虚拟环境打包提交）

由于条件限制，将几个部分在同一台电脑上运行，只是以不同的进程运行在不同端口上，若要分布在不同机器上，只需在 classify.py 中修改集群信息 cluster 的配置将不同部分的地址改为对应机器的 ip 和端口即可。）

二、 系统架构介绍

1. 组件与组件间通信

1.1 组件介绍

分布式系统由 client、master、parameterServer 和多台 worker 组成。各部分主要功能如下：

Client	负责计算图的构造（简化为参数和训练方法的定义），根据集群配置信息连接 master 并与 Master 进行通信，将计算图的定义和集群配置信息传递给 master。通过调用 client.train 过程开始整个模型的训练。
Master	接收 client 传送的配置信息，初始化与 parameterServer 和 workers 的连接，接收计算图的定义并将其注册到 workers 上。Train 过程中将训练数据分配到不同 worker 上。
ParameterServer	与 workers 通信，在每次迭代中将参数发送给各个 worker，并在接收各个 worker 传送的参数更新梯度后，进行参数的更新。
Worker	负责主要的计算过程，接收 ParameterServer 传输的参数，并在接收到 master 传送的训练数据后按照反向传播算法进行参数更新梯度的计算，将计算结果发送给 ParameterServer。Worker 间互不通信。

1.2 组件间通信

各部分间的通信如下：

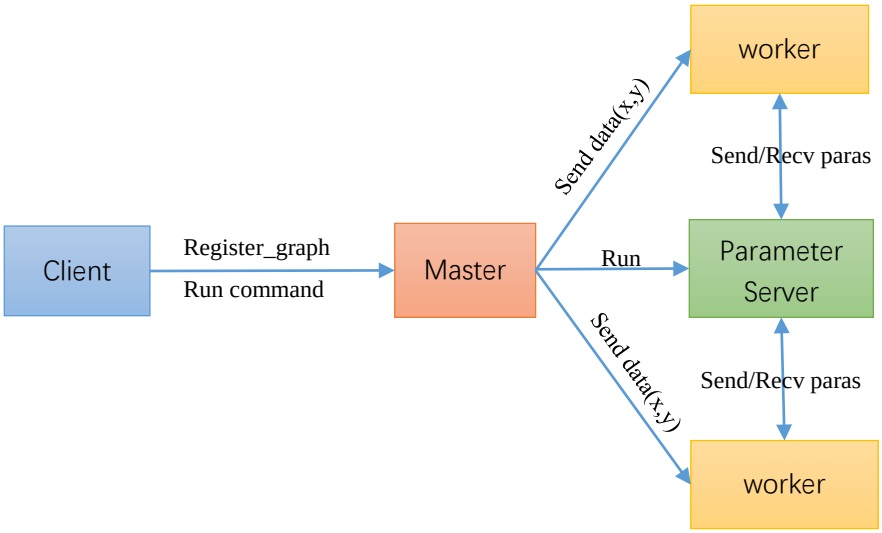


图 1.2 组件间通信

2. 执行流程

该分布式系统运行时，首先需要将各服务端在不同进程中运行起来，并在不同的端口上等待连接。

具体的项目在使用该系统时，需要实例化 Client 类（这里的 client 相当于 TensorFlow 中的 Session），创建一个可以请求服务的 client 端，即调用 client = Client（）。之后训练时只调用 client 的方法，而 client 会负责建立起分布式组件间的连接，并通过 grpc 远程调用服务端的方法进行分布式训练。

组件间通过远程调用的方式进行通信。这使得在每一个组件上，执行的过程都类似于本地调用。

要进行训练，client 端需要先注册集群（register cluster）以建立起各组件间的连接；注册成功后注册计算图（register graph）以定义各 worker 上的计算过程和参数的初始化。之后通过读入数据并迭代执行训练步骤（Train）开始训练。

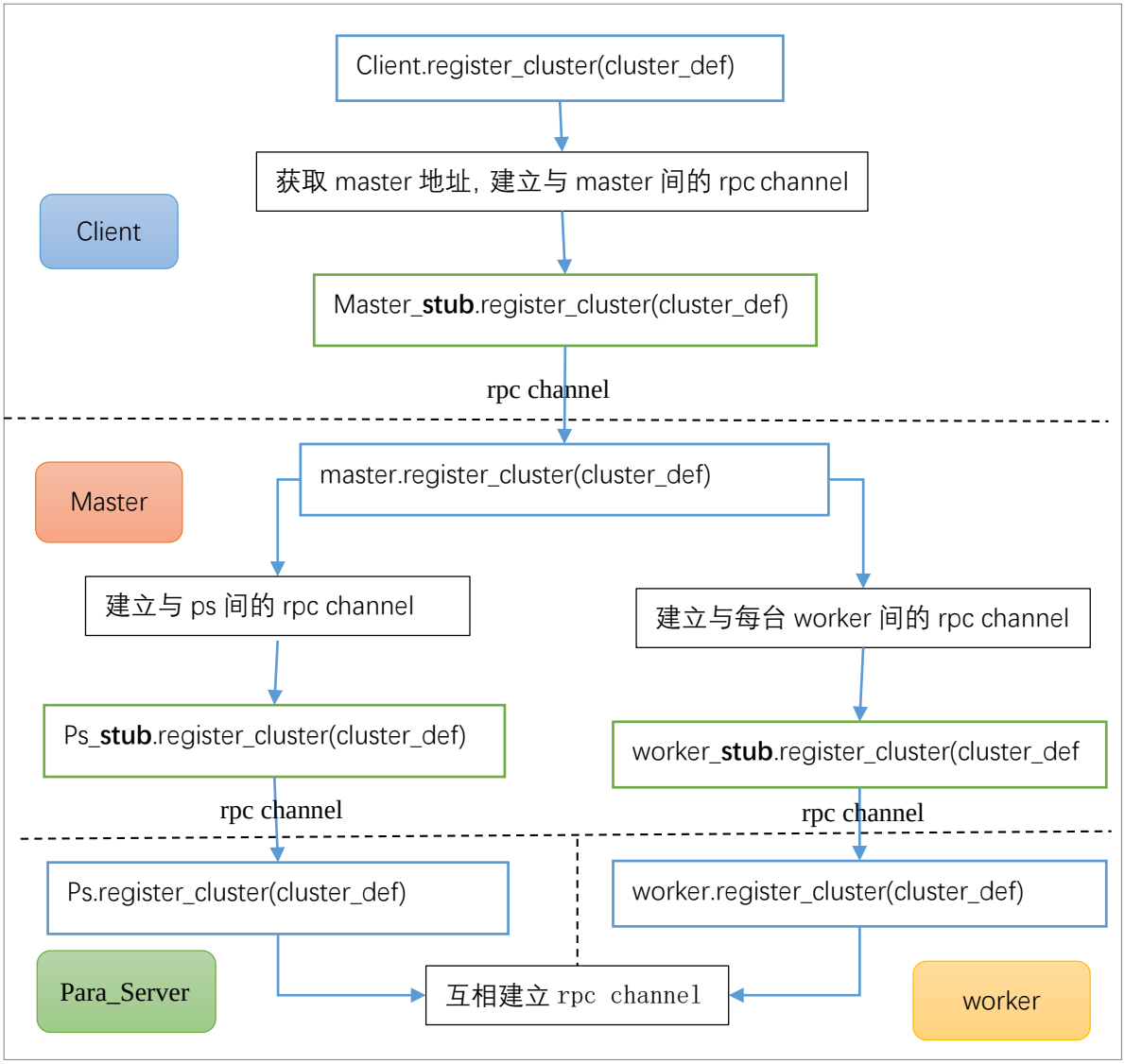
2.1 Register cluster

由于系统中的集群可动态配置，故分布式系统运行之前，各组件需要知道其他组件的地址并建立起连接。

集群的以 map(字典)的形式进行配置，需要制定 master，ps(parameterServer)以及 workers 的 ip 和端口。其中 workers 为数组，可以配置多台 worker。配置示例如下：

```
cluster = {"master": "localhost:2222",
           "workers": ["localhost:2223"],
           "ps": "localhost:2224"}
```

定义好集群配置 map 后，通过调用 *client.register_cluster(cluster_def)*来初始化集群。执行流程如下：

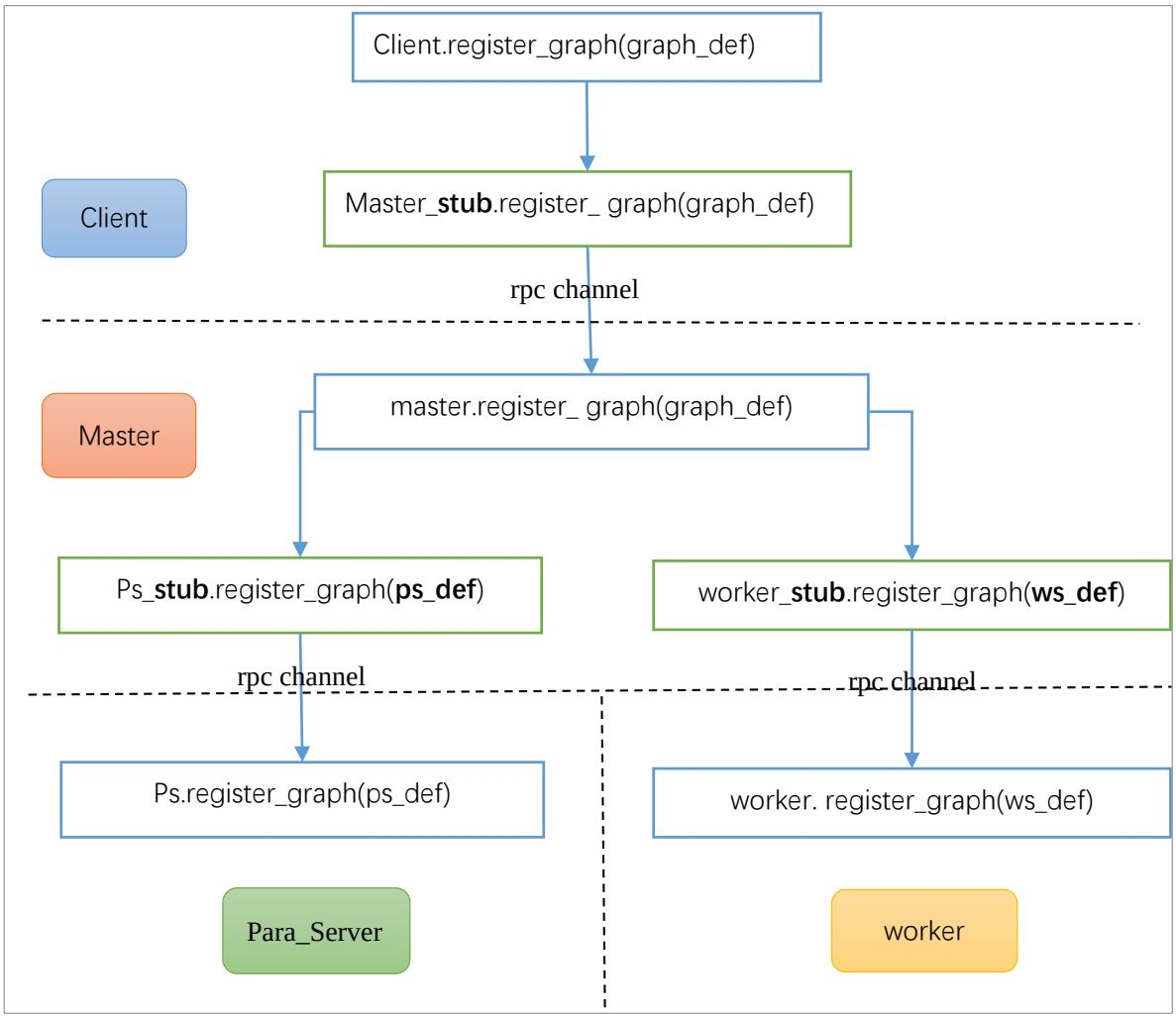


2.2 Register graph

在机器学习中，计算图是训练的核心，在具体项目中定义计算过程后，client 会根据计算过程构造计算图，将其传递给 master，由 master 剪枝处理后注册到 ps 和 worker 上。这里由于计算图的实现过于复杂，便在 worker 和 ps 上提前注册好计算子图，而将图的传递过程简化为初始化参数和训练所选择的损失函数，激活函数的传递，以模拟注册计算图的过程。

系统中可以定义的参数有 W(权重) b（偏移量）c(调节步长)，active_fuc(激活函数，可设置为“sigmoid”或“softmax”)，loss_fuc(损失函数，可设置为“cross_entropy”或“square_error”),client 将所有参数注册到 master 中，master 将其拆分，将可训练参数（w、b、c）注册到 ps 中，函数选择注册到每一台 worker 中。至此，分布式系统的初始化完成，可以等待数据开始训练。

注册图过程如下：

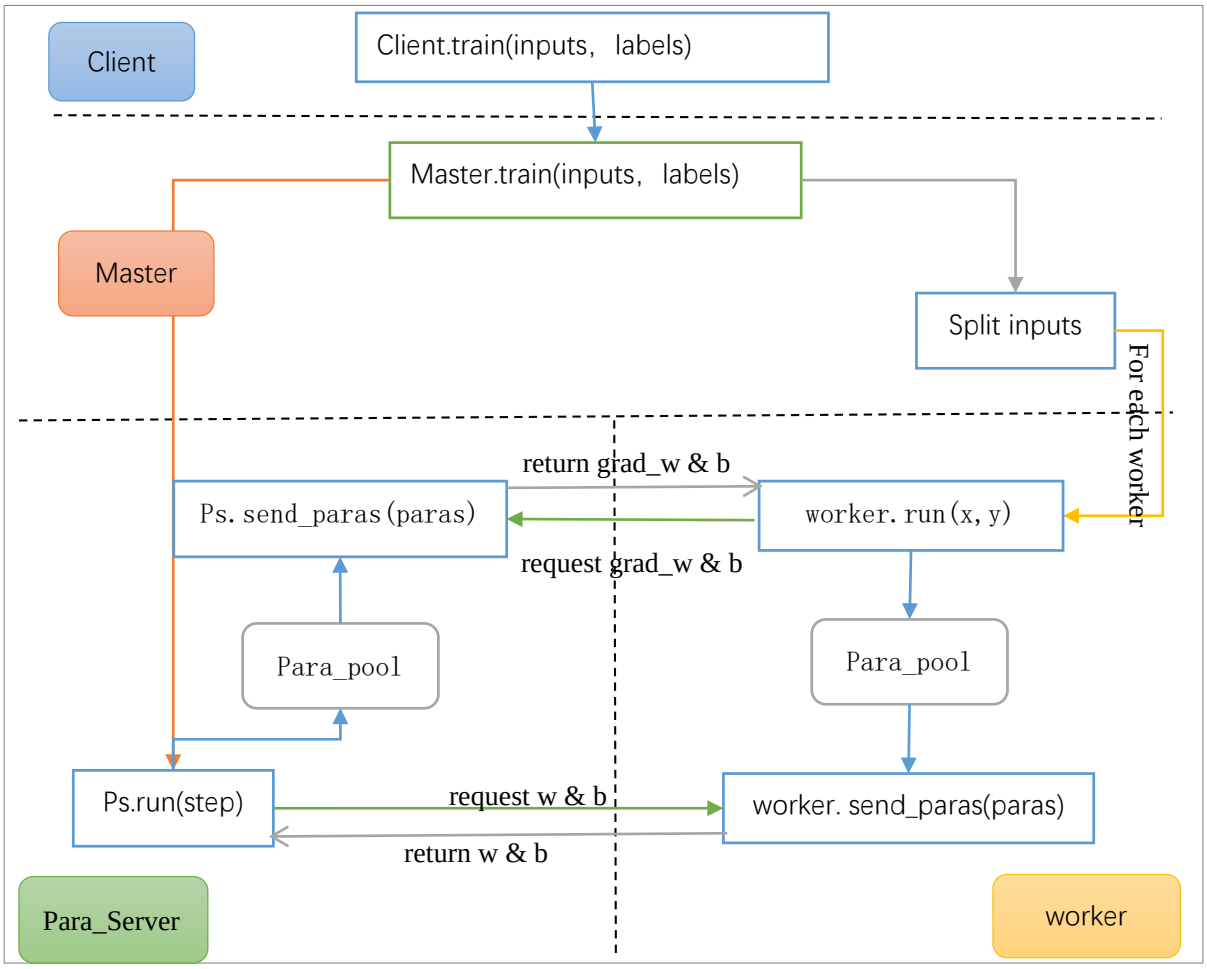


2.3 Train

训练中采取 mini_batch 的方式进行训练，如 batch_size 为 30，则一次性输入 30 组数据（inputs， labels），其中 inputs 为 30 * 3 的数组，每条数据的输入含 3 个维度，labels 为 30*2 的数组，每条数据表示输入对应的输出，如[0,1]表示结果为 1，[1,0]表示结果为 0。

获取输入数据后，调用 client.train(inputs,labels),client 会远程调用 master.train， master 根据 worker 的数量，将数据分为几个部分，调用 worker.run(inputs,labels)使 worker 获得数据并开始计算，worker 向 ps 发送请求获取参数，worker 计算出结果后将结果发送到自身的参数池中。Master 同时通知 ps run，Ps 运行时异步请求每一个 worker 的计算结果，所有 worker 返回后，计算平均梯度以更新参数，并将参数发送到自身的参数池中，供 worker 下次请求。至此，一次迭代训练完成。

流程如下（远程调用同上通过 stub 实现，下图中省略此步骤）：



三、 实现说明

1. 组件与组件间通信

在分布式系统中，客户端与服务端，以及服务端的 master 与 parameterServer, Worker 间都需要进行通信来完成请求、响应以及数据的传输。在本系统中，采取了 google 的开源 RPC 框架 grpc 来实现远程调用，以完成通信。

Grpc 采用 protobuffer 来格式化请求的参数，protoBuffer 是一种高效的序列化技术，客户端将请求及其参数通过 protoBuffer 序列化，发送到服务端后再进行反序列化以完成请求，响应也以同样的方式发送回去。

定义好请求与响应的 proto 文件后，可以编译生成对应的 stub，其中生成的_pb2.py 定义了可装载的参数，_pb2_grpc.py 文件定义了服务端需要实现的服务类和可供远程调用的方法。（proto 文件很好的展现了几个组件的方法和参数）

如 client 需要请求 Master 的服务，将 Master 服务类定义为：

```
service master{//表示Master服务类，定义其可供远程调用的方法
    rpc register_cluster(cluster)returns(status);//初始化集群，建立与ps,worker间连接
    rpc register_graph(graph_def)returns(status);//请求将图分别注册到ps、worker中
    rpc train(data)returns(res);//请求master执行一次训练过程
}

message cluster{//初始化集群的可装配参数
    string ps = 1;//ps地址
    repeated string workers = 2;//数组，workers地址
}
message status{//响应参数
    int32 code = 1;//状态码
    string mes = 2;//响应信息
}
```

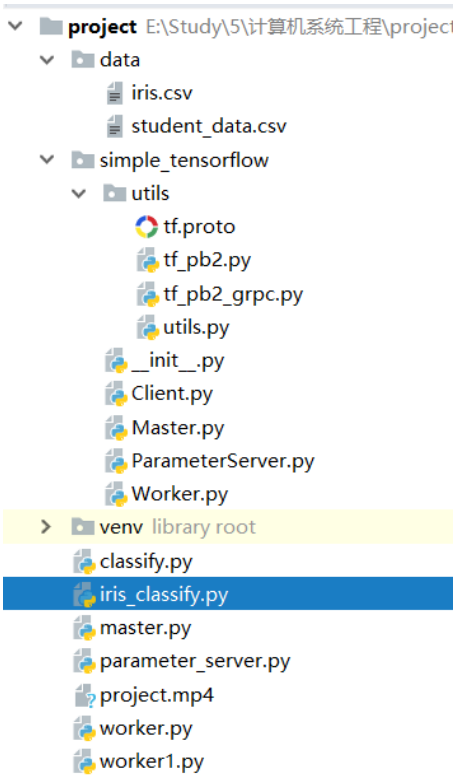
实现服务类后，client 端通过 master 端地址创建 grpc 通道，并以此通道获取 masterStub 桩，通过此桩即可直接调用 master 上的方法，完成请求：

```
conn = grpc.insecure_channel(addr)  # 创建grpc通道
self.master = tf_pb2_grpc.masterStub(channel=conn)  # 获取master桩
status = self.master.register_cluster(  # 请求register_cluster方法并获取响应
    tf_pb2.cluster(ps=cluster_def['ps'], workers=cluster_def['workers']))
print(status.mes)
```

2. 类与实现

2.1 项目结构

整个项目的目录如右图：simple_tensorflow 为分布式系统核心实现部分，其中的 Client.py 、 Master.py, ParameterServer.py、Worker.py 分别实现了上述介绍中的四大组件；utils 目录下为定义的 proto 文件以及编译生成的定义装载参数和桩的 python 文件；utils.py 中定义了各类中需要用到的一些工具函数。



与 simple_tensorflow 同级目录下的其他文件用于调用 simple_tensorflow 以实现具体的机器学习项目，如此次的分析处理 student_data.csv 中数据以进行训练的项目，和鸢尾花分类项目。

2.2 simple_tensorflow

几个分布式组件的类与方法定义如下，具体作用在第二部分的执行流程中有所体现：

```
service master{//表示Master服务类，定义其可供远程调用的方法
    rpc register_cluster(cluster)returns(status);//初始化集群，建立与ps,worker间连接
    rpc register_graph(graph_def)returns(status);//请求将图分别注册到ps、worker中
    rpc train(data)returns(res);//请求master执行一次训练过程
}

service ps{//ParameterServer服务类
    rpc register_cluster(cluster)returns(status);//初始化集群，建立与workers间连接
    rpc register_graph(ps_def)returns(status);//注册图，即初始化参数
    rpc run(step)returns(status);//向各worker请求计算结果，更新参数并将参数发送到自身para_pool中
    rpc send_paras(step)returns(paras);//接受worker请求，在para_pool中取参数返回给worker
}

service worker{
    rpc register_cluster (cluster) returns (status);//初始化集群，建立与ps间连接
    rpc register_graph (ws_def) returns (status);//注册图，即选择激活函数和损失函数
    rpc run (step_data) returns (res); //执行计算过程，计算后将结果发送到本地para_pool中
    rpc send_paras (step) returns (paras);//接受ps请求，在para_pool中取计算结果返回给ps
}
```

Utils.py 中实现的方法与作用如下：

connect_master(addr)	建立与 addr 的 rpc 通道，并基于此通道创建 master 桩并返回。用于 client 获取 master 桩。 Connect_worker 与 connect_ps 类似。
to_array_proto(value)	将任意维的数组转换为 proto 文件中定义的格式。由于 proto 文件不能直接表示多维数组，故需要设计函数进行转换。
to_value(proto)	将 proto 文件中的数组格式转换会多维数组。
gene_data_iter(inputs, labels)	给定多组数据，生成数据迭代器。用于 grpc 中的流式传输，即从 master 传输数据到 worker 时，不用等所有数据全部传输完才开始执行计算，而是以流的形式发送数据，接收到部分数据即可开始计算，以提高效率。
softmax(x)	激活函数
sigmoid(x)	激活函数
compute_grad(x, y, w, b)	正向计算结果并基于 cross_entropy 反向传播计算参数下降梯度并返回。
compute_grad_s(x, y, w, b)	正向计算结果并基于 square_error 反向传播计算参数下降梯度并返回。

2.3 测试项目实现

为测试该系统，提供了测试案例——利用该系统对 student_data.csv 文件进行数据读取训练，通过 GRE Scores (gre), GPA Scores (gpa), Class rank (rank)数据预测能否被录取。

其中 classify.py 实例化了客户端 client，该文件定义了含有一台 master，两台 worker，一台 parameterServer 的集群，均运行在本地，端口为 2222 - 2225。由于输入层有三个节点，输出层为两个节点，故权重初始化为 3 * 2 的 0 矩阵，b 定义为长度为 2 的向量，训练步长 0.001，激活函数 softmax，损失函数为交叉熵。Load_data（filename）函数将 csv 文件读取成 inputs 和 labels 数组，用于训练。

则客户端将整个训练过程定义为：

```
client = stf.Client()
client.register_cluster(cluster)
client.register_graph(w, b, c, active_fuc="softmax", loss_fuc="cross")
for j in range(10):
    for i in range(10):
        print(client.train(train_inputs[i*30:30*i+30], train_labels[i*30:30*i+30]))
print(client.train(test_inputs, test_labels))
```

对于服务端，只需要调用 simple_tensorflow 并将对于服务绑定到 grpcServer 上，在指定端口上运行服务即可。

如 master 上核心部分即为 serve 函数，运行该函数即可提供服务。parameterServer 和 worker 只需将注册服务改为 register_ps（grpcServer）和 register_worker(grpcServer)即可。

```
def serve():
    grpcServer = grpc.server(futures.ThreadPoolExecutor(max_workers=4))  # 创建grpcServer
    stf.register_master(grpcServer)  # 注册master服务到该grpcServer上
    grpcServer.add_insecure_port('[::]:{}'.format(_PORT)) #绑定端口
    grpcServer.start()  # 开始运行服务
    print("start server at port {} ...".format(_PORT))
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        grpcServer.stop(0)
```

四、 设计说明

1. 设计原理

1.1 为什么将系统划分为四个模块

该系统提供了一个低低配版的类似 TensorFlow 的分布式机器学习框架。在 TensorFlow 中，支持通过配置集群将训练过程分布到多台服务器上的多个 cpu 以及 gpu 上，以提高训练速率。

之所以采用分布式系统，是由于在训练数据量大，计算过程复杂的情况下，一方面通过将总的计算分配到不同 worker 上来减小每台机器的训练复杂度，另一方面可以让多台 worker 对不同的数据并行进行计算来加快训练速度。而多台 worker 的情况下，将参数单独放在一台服务器上可以协调多台 worker 计算出结果后更新参数的过程，保障参数更新的数据一致性，且避免 worker 间相互传递参数的过程，使得各台服务器各司其职。同时，需要一台单独的 master 服务器来协调和分配 workers 间的工作，控制 ps 和 worker 间的运行。

故系统中采用的即为 client 与 master 通信，master 与 ps 和 workers 进行通信，ps 和 worker 相互通信，传递参数而 workers 间互不通信的整体架构。

1.2 为什么选用 grpc 实现 Client/Server 模型

Grpc 为 google 开源的采用 protobuf 做协议定义，基于 HTTP2 进行通信的一种 RPC 框架。它的优点在于框架简化了服务的定义，用 protobuf 语言实现了高效地将请求和参数的序列化与反序列化；同时 grpc 是语言无关，平台无关的，它支持客户端和服务端用不同语言和平台实现；另外，grpc 支持流式调用和同步、异步调用，使得分布式系统间的通信更加便捷多样化。Grpc 也是 TensorFlow 分布式框架中采用的 rpc 框架。

RPC 即为远程过程调用，是一种成熟的客户端/服务端编程模型，它隐藏了底层的通讯细节，使得开发时不用直接处理 socket 请求或 http 情求。在使用形式上类似于本地调用，但用它实现的客户端/服务端编程模型与本地调用也有所不

同，可以在系统间实现强制性的模块化。

通过 rpc 定义服务端与客户端的请求，一方面实现了系统的模块化，降低了整个系统的复杂度，比如在分布式机器学习中，worker 上仅实现反向传播算法计算调整梯度，ps 则只关注参数更新策略，调试时更加方便；另一方面使得每一个模块都拥有可替代性，更好地控制系统更改带来的复杂度。比如目前的 worker 只实现了最简单的反向传播算法，当系统需要修改或增加算法时，只需要在 worker 部分进行增改即可；同时，采用远程调用的方式使得分布式和本地式拥有一致性，若不采用分布式，只需要在本地实现对应的服务端的类与函数，client 调用的时候采取本地调用即可。

另外，以 grpc 的方式实现分布式系统也降低了模块间的“同命效应”。Grpc 在远程调用时以 proto 语言来定义参数，这使得模块间的交互过程被限定为指定的类型，非法类型的调用不被允许，这也将错误限制在了传递的消息中。在本系统中，定义的消息类型多为 string 或者 float 等基本类型，一般不会导致系统受到攻击，也避免了编程带来的返回地址错误等问题。同时，也在系统中考虑了异常的处理来减少同命效应。

1.3 为什么在 workers 和 ps 间实现双向异步通信

在整个系统中，参数的训练和更新分布在 workers 和 ps 上，两个模块间需要相互传递参数，ps 需要向 worker 发送训练参数，而 worker 计算完成后需要向 ps 发送计算出的参数梯度，且 ps 需要根据多个 worker 计算出的结果更新平均参数，这使得在这一过程中需要考虑通信的顺序和数据的一致性。在系统中采取的实现方式为：

- master 同时向 ps 和 worker 发送 run 的指令，其中发给 workers 的指令带有训练数据。
- Worker 收到请求后开始执行计算，当计算需要参数时向 ps 发送请求从 ps 的参数池中获得参数，计算完成后将计算结果发送到自己的参数池中。
- Ps 运行时，需要根据各个 worker 的计算结果进行更新计算，需要参数梯度时向各个 worker 发送异步请求从 worker 的参数池中获取参数梯度（当 worker 尚未计算完成时，堵塞操作直到 worker 向参数池中发送参数）。由于

是异步请求，所以可以同时向多个 worker 发送请求，所有 worker 返回后，计算出更新结果，并把更新出的参数发送到自己的参数池中，供 worker 下一次请求。

该设计的好处在于 master 以统一的方式通知 ps 和 worker 同时运行，没有严格的顺序要求，而是用异步通信和阻塞操作保证了参数发送与接收的合理顺序。同时多个 worker 可以同时工作，且 worker 计算参数梯度和 ps 更新参数之间的依赖相对降低了，比如当模型训练好后进行预测时，可以只运行 worker 计算出预测结果而不再更新参数。且降低了节点宕掉后的影响，如在 worker 获取到参数执行计算的过程中，ps 突然挂掉，worker 仍能正常工作，ps 恢复服务后仍能获取到 worker 计算出的参数梯度以继续执行。

2. 系统优缺点

2.1 优点

- **可拓展性强。**该系统以框架的形式提供，没有局限于实现 student_data 的分类任务，而是可以利用该分布式框架实现不同的分类任务。且后续可拓展更多的机器学习算法。
- **实现分布式多 worker 以提高效率。**在文档描述的要求下实现了多个 worker 同时计算，共同更新参数的架构，且 worker 的数目和个服务端的地址可以通过动态配置集群来确定。
- **采用过程调用实现了强制模块化。**如设计原理中所介绍，采用 grpc 使得远程调用如本地调用一般方便，同时将错误限制在指定的消息类型中，并采用异常处理等手段减低了模块间的同命效应。
- **调用简单，结构清晰。**分布式框架以包的形式提供，导入该包后可以以统一地方式实现任意多个服务端，简单的实例化客户端后即可运行。
- **考虑了数据的一致性。**在 ps 和 worker 的通信中，由于有多个 worker 且参数更新后下一次需请求最新的参数，故实现了参数池并以异步、堵塞等方式控制了数据的读取与更新。

2.2 缺点

- **当计算过于简单时，系统间通信的损耗可能大于分布式带来的效率提升**。比如在测试实例中，worker 只执行简单的计算过程且总的数据量小，如果只实现一个 worker，则整个系统与单台机器上顺序执行相比，增加了系统间通信的损耗，而没有带来性能的提升。
- **没有考虑全面节点宕机的情况。**在分布式系统中，整个系统的稳健性、可靠性都是需要考虑的。但在这次设计中，因为节点数目少，所以没有考虑一个节点失效后，用其他节点替换等策略。尽管在没有数据传输时，节点重启等对整个系统没有影响，但当在请求过程中时，节点挂掉会使通信中的节点出现异常。

五、 问题与解决方案

1. 数组参数传递

在设计过程中，采用了 grpc 的远程调用框架，该框架的系统间通信是基于 protobuf 实现的，而 proto 对我来说也是一门陌生的语言。故在研究编写和编译 proto 文件上投入了较多时间。

另一方面，尽管 protobuf 语言定义了多种基本的数据结构且在多种编程语言中都适用，但开发过程中参数 W 可以为多维的数组，而要使系统拓展性强，适用于多种机器学习项目，w 的维度和大小均不固定，故遇到问题：任意维数组的参数该如何传递？

思考后决定利用 numpy 将 w 降为一维，同时获取它原本的 shape（即维度和长度信息），一起作为参数传递。即将 array 的 proto 定义为：

```
message array{
    repeated float value = 1;  //降为1维的数组
    repeated int32 shape = 2;  //数组的shape
}
```

同时在 utils.py 文件中提供了 value 与 proto 进行转换的函数，用于快捷将数组参数序列化与反序列化。

2. PS 与 worker 间的通信

由于一开始实现时，client 与 master 间，以及 master 与 ps、worker 间都采取的是单向通信，即一个为客户端发送请求，服务端响应请求。而在 ps 与 worker 间，ps 需要向 worker 发送训练参数，而 worker 计算完成后需要向 ps 发送计算出的参数梯度。考虑的方案有如下几种：

2.1 实现为 ps 到 worker 的单向同步通信：

ps 向 worker 发送参数后，等待 worker 执行完成后返回计算结果，ps 再进行参数更新。

此方案的问题在于并没有体现出分布式系统的效率。在此方案下，ps 和 worker 间实际为顺序执行，相对与单机的模式来说，没有提高效率反而由于系统间的通信增加了消耗。且在这种情况下，没有较好的方法使多台 worker 同时计算，不能很好地体现出分布式系统的优势。

2.2 实现为 ps 到 worker 的单向异步通信

与上一方案类似，只是采取异步返回的模式，即 ps 发送参数后继续执行其他操作，获取到 worker 异步返回的数据后再进行梯度更新。

该方案能够使多台 worker 同时工作，即向各台 worker 发送参数后，等所有 worker 返回数据后，再更新参数。

此方案问题在于 worker 的计算需要由 ps 触发，而不是由 master 请求运行，且在 ps 请求执行操作时必须保证 worker 已经获取到 master 发送的新的训练数据。这使得 master 上必须先向 worker 发送数据，再向 ps 发送更新参数的命令，ps 在更新参数过程中再请求 worker 计算。三个组件间的调用有顺序要求且依赖性强。

2.3 实现为 ps 到 worker 的双向异步通信

ps 和 worker 同时作为客户端和服务端，分别有 ps 向 worker 发送请求的通道和 worker 向 ps 发送请求的通道。 且采用一个参数池来控制参数的发送与接收。 （该方案的执行流程在设计说明中有所介绍）

该方案的好处在于 master 以统一的方式通知 ps 和 worker 同时运行，没有严格的顺序要求，而是用异步通信和阻塞操作保证了参数发送与接收的合理顺序。同时多个 worker 可以同时工作，且 worker 计算参数梯度和 ps 更新参数之间的依赖相对降低了，比如当模型训练好后进行预测时，可以只运行 worker 计算出预测结果而不再更新参数。且降低了节点宕掉后的影响，如在 worker 获取到参数执行计算的过程中，ps 突然挂掉，worker 仍能正常工作，ps 恢复服务后仍能获取到 worker 计算出的参数梯度以继续执行。

故比较后，最终采用第三种方式：ps 到 worker 的双向异步通信。

3. 计算图与训练方法

3.1 计算图

一开始设计系统时，计算图如何表示以及怎样传递计算图、实现图剪枝、注册子图成了最大的难点，在研究了 TensorFlow 的源码以及一些 TensorFlow 源码解读的文献后，设计了类似的 Variable、Constant、Placeholder 以及 operation 类，并用 protobuf 的形式实现了图的传递。

但是！在实现根据任意的计算图进行公共子表达式消除和子图分裂的时候，数据结构的知识受到了很大的挑战（o(╥﹏╥)o 一把辛酸泪啊……）。折腾了几天后惊觉这是计算机系统工程的 pj 而不是数据结构的…

于是向助教小姐姐求证后采取了简易的方式定义计算图，即在工具类中实现好反向传播算法，在 worker 和 ps 上定义好需要执行的计算步骤，将可拓展的部分缩减为可定义初始化参数和选择仅有的训练函数。

3.2 训练方法

在训练方法上采取了没有隐含层的反向传播，实现二分类问题，但在训练过程中发现每次训练，全数据集上迭代一次，就达到了 68%左右的准确率，但之后的训练准确率没有变，在测试集上的准确率始终是 65%左右。一开始以为是自己的算法写错了… 但用 TensorFlow 实现反向传播训练得到的是同样的结果。观察数据集后发现可能是数据集本身规律性较差，用单层的反向传播只能达到这样的效果。

为了检验系统是否真的成功进行了训练，在网上找了经典的鸢尾花分类数据集，该数据集采用四个维度的输入数据，输出为 3 种鸢尾花类型，数据以 iris.csv 格式提供。基于此任务构建分布式机器学习系统与上一种数据集类似，服务端的配置完全一样，client 的实例化只需要将参数 w 设置为[4,3],b 的长度设置为[3]即可，数据文件读取的方式作出相应调整后即可开始训练。

训练结果如图：

```
cycle 0 : accuracy 0.30833333333333335
cycle 1 : accuracy 0.3416666666666667
cycle 2 : accuracy 0.5583333333333333
cycle 3 : accuracy 0.6416666666666667
cycle 4 : accuracy 0.6416666666666667
cycle 5 : accuracy 0.6416666666666667
```

训练 1000 次后，测试集精度能达到 82.75%。由于数据量较少且网络结构简单，准确率没有太高，但可以看出分布式机器学习系统已在正确工作。

且可以看出系统的易用性和可扩展性，对于不同的分类问题都可以简单的实现。

```
cycle 993 : accuracy 0.725
cycle 994 : accuracy 0.725
cycle 995 : accuracy 0.725
cycle 996 : accuracy 0.725
cycle 997 : accuracy 0.725
cycle 998 : accuracy 0.725
cycle 999 : accuracy 0.725
[[1.0, 0.0, 0.0], [1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [1.0, 0.0, 0.0],
predict accuracy: 0.8275862068965517
```