**关于Delta相关调研**

目录

[横向联邦学习 2](#_Toc135062169)

[Delta计算框架介绍 3](#_Toc135062170)

[Delta Node介绍 4](#_Toc135062171)

[去区块链 5](#_Toc135062172)

[内网脱机部署 7](#_Toc135062173)

# 横向联邦学习

在Google最早的输入法应用中，为了使用用户在输入法中输入的全部历史数据，预测用户将要输入的下一个词，并推荐给用户，以提高用户打字的速度，但是常规的机器学习算法要求用户将本地数据进行上传，以供服务器进行相关处理，但仅仅对于输入法应用这一简单应用我们就会发现，用户用输入法打出的全部历史数据显然是非常私密的，没有人会想要将这些数据发送出去，让任何其他人看到。但是要提高下一个词的预测的准确度，就必须要有很多很多人的输入历史，一起来训练一个机器学习模型。数据越多，预测越准。

在这里我们考虑采用横向联邦学习的方式进行数据的处理和预训练，使得每个用户的数据都可以在本地作为数据集进行训练，其实现原理就是将机器学习的模型部署到每个用户的终端上，使用用户本地的数据对模型进行训练，再聚合每个用户训练得到的模型，用来更新全局的模型。

横向联邦学习中包含两种角色，服务器与客户端。服务器在逻辑上是唯一的，客户端会有许多个。

服务器：保存、更新全局的机器学习模型，协调、配置各个客户端对模型进行训练。

客户端：保存自己产生的隐私数据，使用自己的数据对服务器提供的模型进行训练。

横向联邦学习与传统的机器学习类似，使用梯度下降的方法，对模型的参数进行更新。训练分为很多轮，每一轮都需要:

选取数据：选择一批训练样本

训练模型：使用选取的数据训练模型，得到模型参数的梯度

更新模型：使用梯度下降法，更新模型的参数

不断重复这一过程，直至模型达到收敛。

与传统的机器学习相比，区别在于：

选取数据：数据不储存在服务器上，而是在客户端上，选取数据就是选取一些客户端提供数据

训练模型：模型不在服务器上训练，而是在客户端上进行训练，训练得到的梯度也保存在客户端上

更新模型：模型的梯度分散在各个客户端上，需要对梯度进行聚合，才能进行梯度下降

在整个联邦学习的过程中，我们可以将训练流程分为三个阶段：

**选择阶段**

选择阶段是一个轮次的开始。当客户端满足开始训练的条件（有训练所需的数据）时，会主动与服务器之间建立连接，尝试开始训练。

服务器与客户端建立连接之后，会根据联邦学习的配置，选择接受或是拒绝这一连接。联邦学习中一个轮次中所需要的用户数量有限，并不是越多越好，只要达到了最低的限度即可。服务器会从所有连接的客户端中，随机地挑选出足够的客户端。对于没有被选中的客户端，服务器会向他们发送指令，让其过一段时间再进行连接，然后断开连接。

**训练阶段**

在选择阶段完成后，服务器会将本轮训练需要的配置，发送给客户端。这些配置包括了执行计划、当前模型的参数、聚合的方法等等。在这里，模型的执行计划包括模型结构的定义、如何读取数据、如何计算模型的梯度等。

客户端收到服务器发送的配置之后，会读取本地的数据，运行执行计划，对模型的进行训练，得到模型参数的梯度。

**更新阶段**

客户端得到梯度之后，根据配置的聚合方法，将梯度上传给服务器。

服务器收到客户端上传的梯度之后，会根据配置的聚合方法，对梯度进行聚合，求得所有上传梯度的平均值，根据配置的优化方法，对当前的模型参数进行更新，并持久化，供下一轮使用。

# Delta计算框架介绍

在Delta框架中，节点被称为Delta Node，Delta Node 同时包含横向联邦学习中的服务端和客户端的功能。 同时，在Delta Node之上，还有一层区块链层，通过部署在区块链上的智能合约，来协调各个Delta Node之间的通信与任务执行。如果不需要保证计算结果的可信性，也可以去掉区块链，直接使用Chain Connector来组网以及协调任务。

当一个新的Delta Node连接到Chain Connector，它首先会调用注册节点方法，将自己注册到整个网络中来。注册的时候，包含了自身的地址，这样，其他的Delta Node就能连接到它了。

当某个Delta Node提交了任务，那么它就会调用创建任务方法，在智能合约中创建一个任务，并生成一个创建任务的事件。其他的Delta Node可以订阅这个事件，当有新任务生成时，就能收到事件，这样就能决定是否加入这个任务了。 如果要加入这个任务，那么就可以和创建这个任务的Delta Node进行通信，加入任务，参与训练。

当创建任务的Delta Node节点，开始新的一轮训练时，会调用开始训练方法，并生成一个开始训练的事件。其他的Delta Node可以订阅这个事件，当开始训练时，它们也在本地开始训练，从创建任务的Delta Node那获取任务，进行训练， 上传梯度，完成这一轮训练。

不难发现，Delta可以联合分散在各处的数据，进行统计计算以及机器学习，数据全程不会离开本地，实现了数据的可用不可见，可应用于金融、医疗、政府、知识产权等领域，实现联合风控、联合科研、政企数据连接等需求，充分发挥数据的价值。

Delta通过封装整合联邦学习、安全多方计算、差分隐私等最新的隐私计算技术，降低了开发门槛，使用者无需了解隐私计算技术，也可快速实现计算需求。 用户可以快速地部署Delta节点，搭建隐私计算网络，联合多方数据，完成隐私计算。

Delta保证了计算结果的可信性，在用户无法获取原始数据的前提下，通过集成的区块链和零知识证明技术验证计算结果的正确性。

# Delta Node介绍

Delta Node同时包含了服务端与客户端的功能，即可以作为某个任务的创建者，开启服务端，进行协调；也可以开启客户端，作为某个任务的参与者，使用本地数据进行训练。

在Delta Node的服务端启动时，需要绑定一个地址，以便让其他节点可以访问本节点。

在Delta Node的服务端中，包含了一个HTTP Server，一个gRPC Server，以及一个对接区块链系统的Blockchain Client。

其中，HTTP Server用于对用户提供操作接口，包括：

新建联邦学习任务

查询本地的任务列表

查询任务运行状态

查询任务运行的日志

下载任务的结果（模型权重）

对于用户来说，启动Delta Node后，可以通过调用新建联邦学习任务的接口，新建任务。 新建任务接口在数据库中保存任务的信息（任务名称，所需的数据集，创建者等），在本地磁盘中保存任务的执行计划、模型权重等， 并通过Blockchain Client在区块链中注册任务，注册的任务包含了任务的基本信息以及本节点的地址。注册任务可以获取任务的ID，然后将任务ID返回给创建者。 创建者通过任务ID，就可以通过其他接口，查询任务的执行状态、运行日志、下载结果。

Delta Node中的gRPC Server，主要是用来与其他Delta Node进行通信的。gRPC Server主要包含以下方法：

加入任务

结束任务

获取任务配置

下载任务权重

安全聚合

在Delta Node中，选择使用gRPC Server的主要原因是为了更好地支持流式通信，以方便地实现安全聚合（因为安全聚合算法一般需要长连接与多轮通信）。 当HTTP Server中新建了一个任务之后，其他Delta Node节点的客户端可以调用加入任务方法，加入任务。 加入任务后，其他Delta Node 节点的客户端可以调用获取任务配置、下载任务权重方法，将任务下载到本地，进行训练。 训练到一定轮次后，其他Delta Node 节点的客户端会调用安全聚合方法，根据不同的安全聚合算法配置，进行安全聚合。 安全聚合后，服务端会更新本地保存的任务模型权重，供下一轮中Delta Node节点客户端使用。 当Delta Node节点客户端完成任务训练后，会调用结束任务方法，结束该任务的执行。

对接区块链系统的Blockchain Client，主要功能是调用区块链中部署的智能合约，在区块链中注册任务，以便让其他的Delta Node能够发现任务并加入进来。

在Delta Node的客户端启动时，需要指定一个保存数据集的文件夹，这个文件夹中，包含了本地可以用来训练的数据。

在Delta Node的客户端中，包含了一个对接区块链系统的Blockchain Client，以及任务执行器executor。

对接区块链系统的Blockchain Client，主要功能是监听区块链智能合约中的注册任务事件。当监听到新的任务注册事件后，客户端会生成一个相应的任务， 并送入任务执行器executor的任务队列中，等待执行。

任务执行器executor中会维护一个任务队列，不断从任务队列中取出任务并执行。当收到一个任务时，它会首先检查，是否可以执行这个任务（即是否有这个任务所需的数据）， 如果可以执行，就为任务新开启一个进程，独立执行。

执行开始时，根据任务所对应的服务端Delta Node节点地址，建立连接，尝试加入任务。 如果加入任务失败，表明这一轮训练中，本节点没有被选中，可以过一会再次尝试加入任务。executor会将当前任务放入它的任务队列中，并关闭当前执行的进程。 如果加入任务成功，则会开始下载任务的配置、任务权重，读取本地对应的数据，进行训练。 根据配置，训练一定轮次后，可以开始进行安全聚合，安全地上传本地训练得到的权重或梯度。 安全聚合之后，本轮次就结束了。之后重新尝试加入任务，重复上述的步骤。

# 去区块链

Delta计算框架的主要目的是为了帮助用户高效地进行大规模数据处理和分析，而区块链技术则主要用于去中心化的数据存储和交换，以及保证数据的安全性和可信度。因此，Delta计算框架和区块链技术的应用场景和目的有所不同。

在Delta的设计中，区块链的职责专注于链上结算（审计），只有涉及到交易的验证，相关流程才通过区块链智能合约实现，其他的链下交互，则通过Delta Node之间的二级通信网络进行，这样可以在保证交易可信性的前提下，实现更高的交易处理速度，提高系统可用性。

首先我们先分析区块链合约在整个联邦学习过程中的应用，Delta中的横向联邦学习的过程分为五步：任务注册、参与者选取、密钥交换和秘密分享、本地训练、安全聚合。系统参与者有三方：

数据需求方：也就是计算任务开发者，运行Delta Node的Server端

数据所有者：持有数据，执行隐私计算任务，不暴露自身的数据

Delta区块链：由多个节点组成的区块链网络

设计链上的部分主要有以下几个部分：

在任务注册阶段，任务开发者将计算任务发布到区块链上。这里的核心是发布到区块链上的任务内容。不包括原始的计算代码，或者初始模型参数。链上只存储计算任务的零知识承诺，用于对计算任务的验证，以及后续对计算结果的验证。

任务内容包括了对于数据的需求，使用一个标准化的数据协议进行描述。以及对于节点和数据的一些其他要求。任务还需要设定一个价格，代表自己愿意为这个任务支付多少费用。数据所有者的节点通过这些要求判断自己是否加入一个计算任务。

数据所有者的节点监听区块链事件来获取任务，如果选择加入一个任务，则开启监听另一个计算任务轮次的通道。等待计算轮次的开启。

注意在任务注册阶段，数据所有者并不需要在链上报名参加任务，这是针对大规模横向联邦学习的场景的设计。很多情况下，大规模横向联邦学习的数据持有者都是移动设备，比如手机。而手机是非常容易掉线的，比如手机持有者突然进了地铁，或者是手机没电了。因此我们的流程设计不能假设同一个节点会长时间在线。而同时手机用户又特别多，总会有很多用户在线进行计算。因此Delta中的横向联邦任务以轮次做为计算节点参与的最小单位。每一轮选择一批节点，节点计算完成提交后立即结束，不要求其下一轮仍然在线。

任务执行的基本单位就是轮次，一个横向联邦学习任务，可以有非常非常多的轮次，时间跨度也可以特别长。每个轮次的计算结果叠加到之前的模型上，构成一个最新的训练模型。

任务发起者在链上发出开启轮次的事件，接收到事件的计算节点调用智能合约报名参加，等待一定时间后，报名结束，任务发起者在报名的节点中随机选择一批，参与本轮的计算。

由于计算节点会随时掉线，因此Delta的设计原则是保证单轮运算的时间尽量短。因此这里的等待时间必须严格控制，防止节点加入轮次后掉线。

安全聚合的过程和节点掉线容错的处理被集成到了一起。当区块链发出开启安全聚合的事件后，各个计算节点通过链上的数据判断出哪些节点提交了计算结果，哪些节点掉线了。假如掉线的节点数量在可容忍的范围内，开始执行安全聚合。

对于每个掉线的节点，本计算节点对外发送其用于生成Mask的SK2的秘密分享，用于反推出其Mask并从最终结果中移除。对于每个在线的节点，对外发送其生成随机数的Seed的秘密分享，用于在结果中移除随机数。

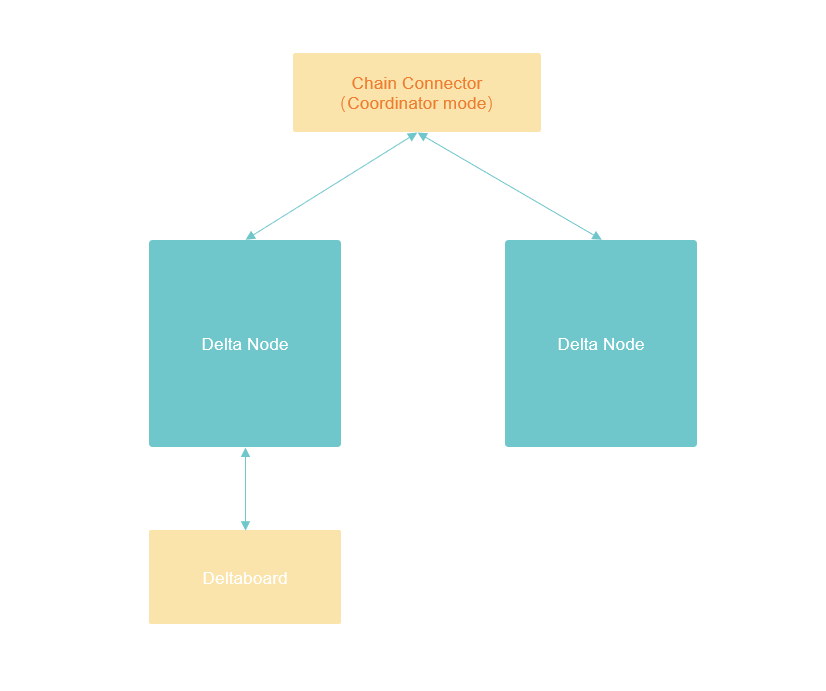
服务器拿到全部的秘密分享后进行相加，得到最终的横向联邦学习的计算结果。

注意这个流程中，计算节点在对外发送秘密分享以前，同样也需要先将秘密分享的零知识承诺发送上链，而服务端接收到秘密分享后，也是需要先根据链上的承诺进行验证。和上一个步骤里的流程非常相似。这个步骤的设计有几个原因，一个是因为目前在链上还无法实现对于复杂零知识承诺的验证，因此需要服务器端链下手动验证，另一个是因为区块链上无法存储大量数据，因此没法通过区块链进行加密后的模型训练结果的直接分发。在未来的实现中，通过完整的零知识证明，以及Layer 2支持的区块链分布式存储，这个步骤可以被完全省略，直接在区块链上执行完成。

至此一个轮次的训练就已经成功结束了，服务器端拿到了最终的平均结果，计算节点未对外暴露任何数据，同时区块链上有足够的数据对这次计算进行验证和结算。

我们可以进行反推得出在不需要对数据进行交易验证的情况下我们是不需要进行区块链智能合约配置的，这里Delta提供了一个解决方案：

由Chain Connector完成组网，协调Delta Node完成计算任务执行。最小化的网络需要搭建一个Chain Connector（运行于Coordinator模式），两个Delta Node，一个Deltaboard，如下图所示：由Chain Connector完成组网，协调Delta Node完成计算任务执行。最小化的网络需要搭建一个Chain Connector（运行于Coordinator模式），两个Delta Node，一个Deltaboard，如下图所示：



# 内网脱机部署

我们首先参考docker的离线部署模式：

在内网环境下，由于无法连接到公共的 Docker Hub，因此需要采取一些额外的步骤来实现 Docker 的脱机部署。

1.在外网环境下，使用 Docker 镜像导出命令将所需的镜像导出为 tar 包。例如，要导出名为 myimage 的镜像，可以使用以下命令：

docker save myimage -o myimage.tar

将 tar 包传输到内网环境中。可以使用 USB 驱动器、CD-ROM 或其他可移动存储设备来完成此操作。

在内网环境中，使用 Docker 镜像导入命令将 tar 包中的镜像导入到 Docker 中。例如，要导入名为 myimage 的镜像，可以使用以下命令：

cssCopy code

docker load -i myimage.tar

2.在内网环境中，使用 Docker Compose 或 Dockerfile 创建和部署容器。您可以使用 Docker Compose 来定义容器之间的关系，并使用 Dockerfile 来定义容器的配置和依赖项。例如，以下是一个简单的 Docker Compose 文件示例：

yamlCopy code

version: '3'

services:

myservice:

image: myimage

ports:

- "8080:80"

运行以下命令来启动容器：

docker-compose up -d

这将启动一个名为 myservice 的容器，并将容器的端口映射到主机的端口 8080 上。

对于Delta计算框架的