Sparse Classification RBM系统设计构架

蒋宇东

August 12, 2013

本文的目标是设计一个可扩展的分布式Sparse Classification RBM， 以支撑在线广告推荐业务的CTR预估。本文会尽量的讨论系统的设计细节，以避免由于设计的疏忽导致工程实现中出现种种的问题。

1 系统简介

该系统通过对检索查询的相关特征进行有监督的学习从而得到在线广告推荐业务的CTR预估模型。由于在线系统的训练数据量巨大（通常为数亿条），特征的维度高，所以必须要考虑构建分布式的训练系统以降低大规模训练所需要的时间。本文第二节介绍单进程Sparse Classification RBM的基本算法；第三节探讨采用DownPour优化算法做多进程RBM Training开发的思路；第四节讨论要训练更大规模的模型，进行优化的方向和基本实现方法；第四节讨论实现该系统所涉及到的工程技术，包括开发语言，开发库等等。

2 单机版Sparse Classification RBM算法介绍

单机版的训练算法采用CD-1训练RBM，训练模型分为Generative Training和Discriminative Training两个过程，通过对两种剃度更新算法加入不同的权重得到Hybrid Training训练模型。模型的输入层中的所有节点与隐含层中的所有节点保持全连接状态，输出层节点为1具有｛0，1｝两种状态，实现对于CTR的预测。模型的输入层进行了Sparse Coding处理，即将输入层的所有节点切分成若干个non-overlap的group，每次输入保证仅有一个输入节点被激活。详细请参考SparseClassificationRBM.

3 分布式框架

本节将会讨论两种设计方案，实现系统在训练阶段的分布式设计。基于Downpour SGD的设计参考了Jeffrey Dean, Greg G.Corrado等人开发的DistBelief框架，该算法提供了异步更新梯度的算法框架，使得RBM的训练过程可以扩展到多台机器。但由于内存的限制，单纯的使用Downpour SGD框架无法学习大规模参数的全连接模型，因此本节还将讨论基于Alex Krizhevsky开发的并行化训练RBM算法。上述两种算法都已经成功的用于Google Brain项目。通过CPU-GPU并行优化，Google使用1600台高性能服务器构建了Billion级别参数的深度学习网络。

3.1 基于Downpour SGD的构架

Downpour SGD的主要目的是将单一RBM的训练过程分布到多台服务器。该算法的主要思想是：将训练数据划分为若干个子集，分布在多个Worker服务器上，在每一个Worker上都运行一个RBM模型的拷贝。模型的参数更新通过存储参数的中央服务器进行，该参数服务器保存了模型所有参数的当前状态。训练阶段，每个Worker分别从中央服务器获取模型当前所属状态的参数，并根据该参数执行min-batch。

计算好更新梯度后，将结果推送回中央服务器。注意，整个的梯度更新过程完全是异步进行。算法的伪代码描述如Algorithm 1所示：

Algorithm 1 DownPourSGDClient($\alpha, n_{fetch}, n_{push}$)

```
PROCEDURE
StartAsyncFetchParam(params)
 1: params <- GetParamsFromParamServer()
PROCEDURE
StartAsyncPushGradients(accruedgradients)
 1: SendGrandientsToParamsServer(accruedgradients)
 2: accruedgradients <- 0
PROCEDURE
main()
 1: while true do
 2:   if (step mod n_fetch) == 0 then
 3:     StartAsyncFetchParam(params)
 4:   end if
 5:   data <- GetNextMinBatch()
 6:   gradient <- ComputeGradient(params, data)
 7:   accruedgradients <- accruedgradients + grdient
 8:   params <- params - α*gradient
 9:   if (step mod n_push) == 0 then
10:     StartAsyncPushGradients(accruedgradients)
11:   end if
12:   step <- step+1
13: end while
```

DownPour SGD较同步SGD的优势在于：对于同步SGD如果一台机器更新失败，整个更新过程将会推迟；但对于Downpour SGD来说，如果一台机器失败，其他机器的工作将不会收到影响。异步更新可能产生更新顺序的乱序，目前没相关的凸优化理论可以解决这个问题。但是，在实际的操作中，异步更新具有很好的稳定性。

3.2 Adagrad Adaptive Learning Rate

采用Adagrad Adaptive Learning Rate 算法可以很大程度上的提升Downpour SGD算法的稳定性，该算法在更新过程中不是使用固定的learning rate，而是对于不同的参数采用不同的learning rate。设$\eta_{i,k}$为第i个参数在第K次迭代时的learning rate，则有 $\eta_{i,k} = \gamma / \sqrt{\sum_{j=1}^{K} \Delta w_i j^2}$ 由于learning rate值涉及到要更新参数本身的历史状态，所以可以在Worker中实现其计算。

3.3 性能分析

分析内存的使用情况。根据RBM算法训练的原理，设min-batch中每次计算所用的数据量为batch size，节点的数据类型为type，可视层节点为view_dim，单台Worker内存总数为total_mem字节。则可以计算隐含层的数量

$$hid\_dim = (total\_mem - batchsize * view\_dim * 2 - view\_dim * 3) / (batchsize * 4 + view\_dim * 3 + 7)$$

batch_size = 50, type = 4, view_dim= 10000*type Byte, total_mem = 64*1024*1024*1024 Byte 则可计算隐含层节点数hid_dim=286050 个。由此可以估算出可训练模型的规模。

3.4 模型分片

对于在线广告业务，输入特征的维度比较高，其所需的隐含层节点数量也相应的比较多，因此，我们希望训练具有更多参数的网络。如上一节分析可知，单纯使用downpour SGD算法，由于内存的限制，无法进行更大规模的模型训练，故在本节我们将探讨网络分片的方法。 如图Figure 1所示，对整个训练过程进行四路并行，即将可视层以及隐含层划分为4块，分布到不同的主机上。算法的伪代码如Algorithm 2所示： 大多数的权值都会被更新至少两次（在不同的机器上），由于

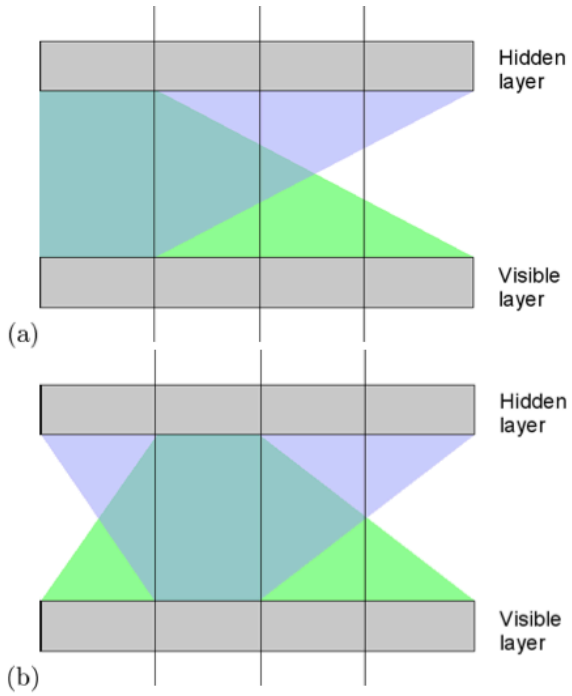

Figure 1: 分布式模型更新图

Algorithm 2 Parallelizing RBM

可视层节点为$V = \{v_1, v_2, v_3, ..., v_I\}$
隐含层节点为$H = \{h_1, h_2, ..., h_J\}$

1: 每台主机需要可以访问整个数据集，所以必须存储可视层节点$V = \{v_1, v_2, v_3, ..., v_I\}$。
2: 主机k，在给定可视层节点V的条件下，计算隐含层节点的状态 $H_k = \{h_0^k, h_1^k, ..., h_{J/K}^k\}$ 。注意，每台机器仅计算J/K个隐含层节点，使用图中绿色部分的连接权值即可完成计算。
3: 所有的机器互相之间通信，将第2步的结果互相传递。
4: 现在每台机器都知道了所有隐含层节点的状态，通过该状态来计算更新可视层(negative data) $V'_{k'} = [v_0^{k'}, v_1^{k'}, ..., v_{I/K}^{k'}]$，此过程仅需要使用紫色部分的权值。
5: 所有的机器互相之间通信，将第四步所得到的结果发送给所有的其它主机。
6: 现在每台机器得到了negative data，再次使用绿色部分的连接权值来更新隐含层节点。
7: 所有的机器互相通信，将第6步的结果互相传递。

浮点数计算精度的不统一问题，导致不同机器对于同一个权值的更新结果会略有差异。这里我们引入权值同步来修正Algorithm 2。权值同步的频度取决于两台机器权值产生差异化的速度。更新步骤如Algorithm 3:

Algorithm 3 Synchronization

1: 将Figure 1中紫色的权重指派为RBM的权重，删除绿色的权重。
2: 机器之间互相传递紫色权值的部分切片以帮助其他机器进行权值同步。

4 框架融合

第三节讨论的两种分布式设计的框架可以进行分别的开发，然后再进行融合，这样可以帮助模型训练更多的参数，更加充分的发挥机器的性能。对于大规模的模型系统，可以首先采用Algorithm 2对模型进行分布，不同的机器将负责不同区域的权重更新。然后可以采用Algorithm 1对单独的一个模型切片进行Downpour SGD优化。