缩略语和关键术语定义

Data batch:每一个训练周期内，系统用来训练的数据块

Sub batch:每一块GPU卡上分到的用于训练的数据，该数据是data batch的子集

BN(batch normalization):批量归一化，指先对一个数据集的所有数据计算均值和标准差，之后对每个数据做归一化操作，减去均值再除以标准差。

1、相关技术背景（背景技术），与本发明或者实用新型最相近似的现有实现方案（现有技术）

1.1 与本发明或者实用新型相关的现有技术*（可通过功能框图、流程图、原理图等并配以文字对现有技术进行说明）*

1.1.1 现有技术的技术方案

目前利用多块GPU卡做DNN的训练模式已经非常普遍，而在此过程中，主要是采用数据并行的技术方案做训练加速。该技术方案如下：在一个训练周期内，系统首先会将拿到的一批训练数据(data batch)按照已有的GPU卡数目，分成相应份数的子集(sub batch)，并分发给每一块GPU卡。在训练时，每一块GPU卡上会加载一套完整的待训练的DNN模型，之后利用分配到的数据去训练当前卡上的模型。在训练完这批数据后，因为训练数据的不同，不同GPU之间训练出的模型权重的梯度会存在差异，这时候会进行模型同步操作，将不同GPU上训练出的梯度进行归约合并，最终得到相同的梯度，再利用最优化的算法去更新每块GPU上的模型权重。之后进入下一个训练周期。基本过程如图1所示：

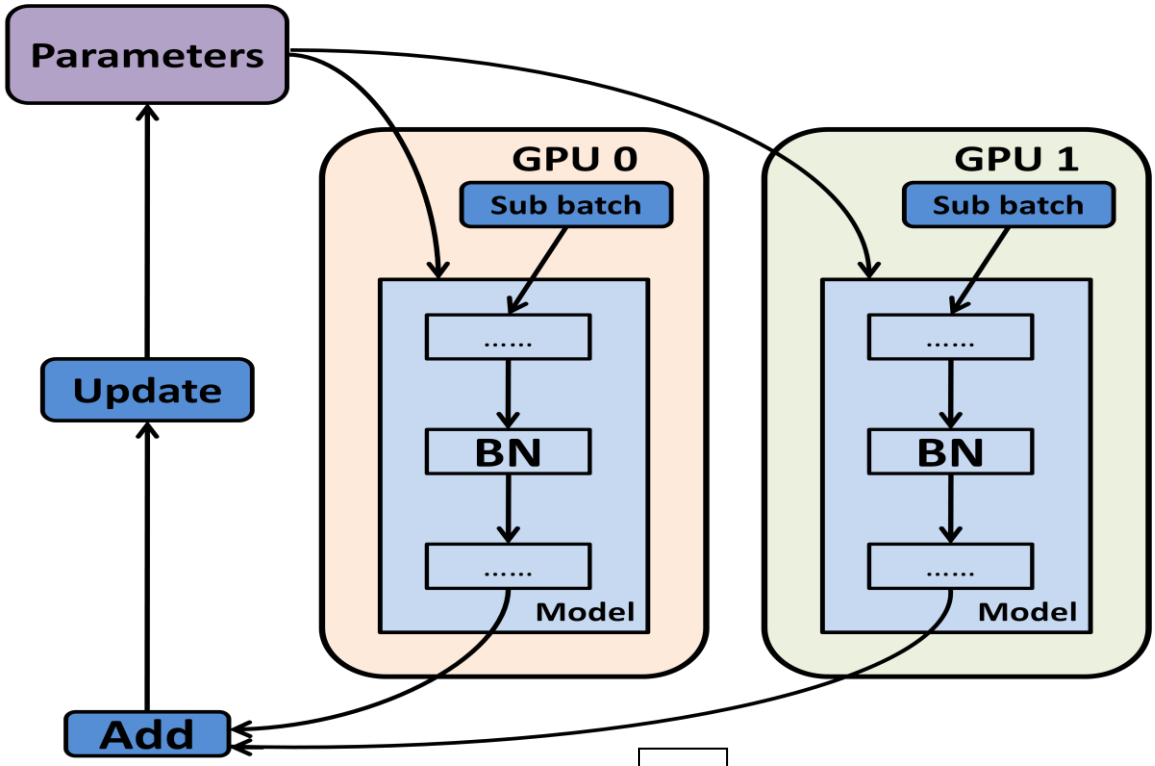


图 1

1.1.2 现有技术的缺陷 *（分析1.1节介绍的现有技术的技术方案的缺陷，该缺陷应是本发明所要解决的技术问题，对于本发明没有解决的现有技术的缺陷可以不写）*

现有技术方案的主要问题在于，将数据分成子集再分发给每块 GPU 卡训练时，每块 GPU 卡因为无法拿到完整的数据集，从而导致数据缺失。在神经网络的批量归一化层(Batch Normalization，以下简称 BN)，数据缺失导致的模型精度下降就会体现出来。这是因为 BN 层的主要操作就是计算全部数据的均值和标准差，之后将每个数据的值减去均值再除以标准差，从而达到归一化的目的。由于每块 GPU 卡上得到的数据不完整，每块卡上计算的均值和标准差会出现偏差，从而影响整体训练的精度。图 2 显示了用数据并行的方式做训练加速时，对模型预测精度的影响。可以看到训练完成之后，3 块 GPU 数据并行的训练(3GPU train)精度比单 GPU(1GPU train)会下降 7%左右，而检验精度(val)平均会下降 15%左右。当 GPU 数量进一步增加时，该精度下降会更加明显。

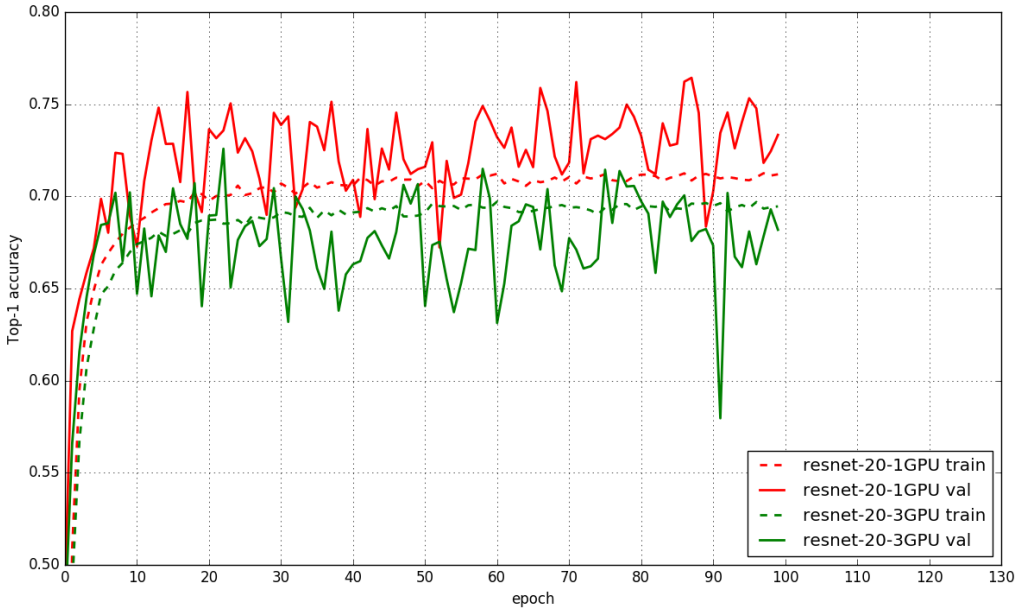


图 2

2、本发明或者实用新型技术方案的详细阐述（发明或者实用新型内容）

2.1 本发明或者实用新型所要解决的技术问题

本发明专注的是如何在多GPU数据并行训练中，提高模型训练的精度，使其达到和做单GPU训练时相同的精度。

2.2 本发明或者实用新型提供的完整技术方案（发明或者实用新型方案）***（方法发明：请提供方法流程图，并结合方法流程图写明该方法包括的每一个步骤，及每个步骤的实现方式。实用新型，请提供产品结构框图，并结合框图撰写产品的组成、各组成的功能或作用以及各组成之间的关系（连接关系、信号传递关系、作用关系等））***

系统概述

我们的系统改进方案如图3所示。与图1相比可以看出，我们的主要改进在于：在前向传播的计算过程中，我们在数据进入BN层之前，增加了预处理(pre-process)操作，用来计算单卡上数据的均值和方差；对该操作结果，我们进行了多GPU的全局归约操作(Reduce)，以计算全局的均值和方差。在反向传播过程中，在数据进入BN层的反向传播节点前，我们添加了完全相同的pre-process和全局reduce操作。由于BN层正向与反向计算过程略有不同，我们对正反向的pre-process和reduce操作进行了相应的修改。

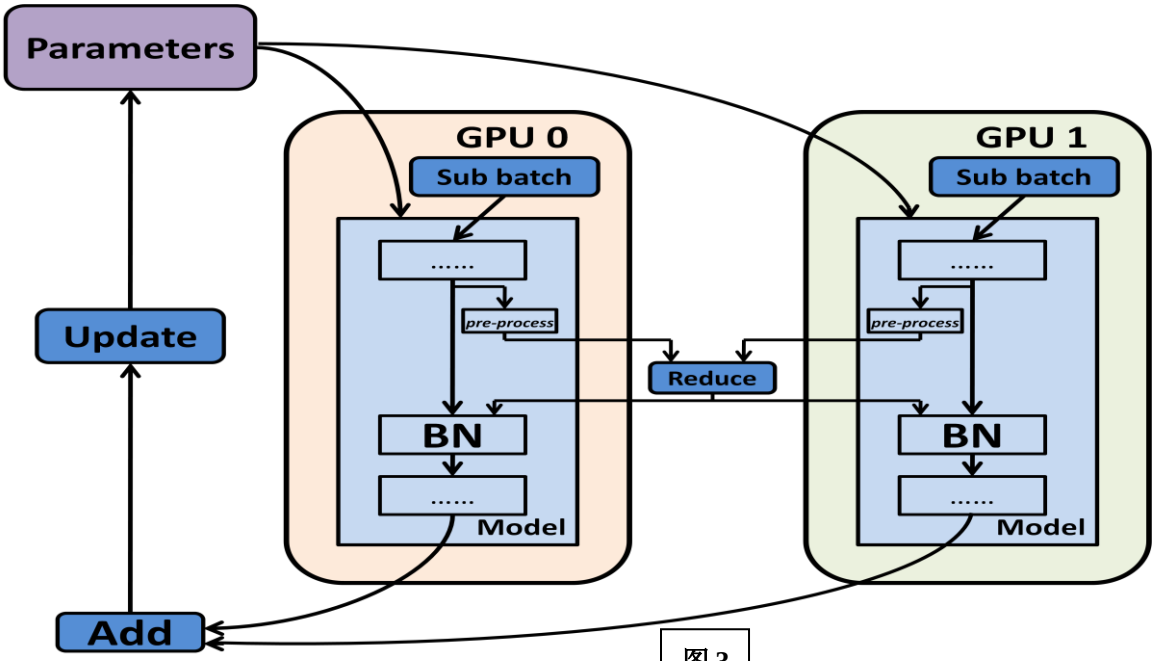


图 3

技术改进方案

（一）正向传播过程的改进

传统的算法中 BN 层的计算过程如下,其中γ,β即为需要训练的参数，m_i是第 i 块 GPU 卡上的数据量

传统 BN 在第 i 块 GPU 卡上的向前传播算法（参考文献[1]）：

第 i 块 GPU 卡上一个 sub batch 的**输入**: $B_i = \{x_{i,j}\}\ (j = 0, 1, 2, \dots m_i)$

参数: γ, β

第 i 块 GPU 卡上的**输出**: $\{y_{i,j} = BN_{\gamma,\beta}(x_{i,j})\}\ (j = 0, 1, 2, \dots m_i)$

计算过程

1：计算均值: $\mu_i = \frac{1}{m_i}\sum_{j=1}^{m_i} x_{i,j}$

2：计算方差: $\sigma_i^2 = \frac{1}{m_i}\sum_{j=1}^{m_i}(x_{i,j} - \mu_i)^2$

3：归一化: $\hat{x}_{i,j} = \frac{x_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}$ （ε 为固定的极小非 0 值，防止除 0 情况发生）

4：计算偏移量: $y_{i,j} = \gamma\hat{x}_{i,j} + \beta$

我们改进后的技术方案将BN分成了预处理(pre-process)，归约(reduce)和归一化(BN)三个步骤，其过程如下：

首先是pre-process过程

算法一: Pre-process 在第 i 块 GPU 卡上的向前传播算法：
第 i 块 GPU 卡上一个 sub batch 的**输入**: $B_i = \{x_{i,j}\}$ $(j = 0, 1, 2, \dots m_i)$
第 i 块 GPU 卡上的**输出**: μ_i, ν_i

计算过程

1：计算均值: $\mu_i = \frac{1}{m_i}\sum_{j=1}^{m_i} x_{i,j}$

2：计算平方的均值: $\nu_i = \frac{1}{m_i}\sum_{j=1}^{m_i} x_{i,j}^2$

其次是全局reduce过程，n是用于训练的GPU卡数目

算法二:全局 reduce 的向前传播算法：
输入: μ_i, ν_i, m_i $(i = 0, 1, 2, \dots n)$
输出: μ, ν

计算过程

1：计算全局均值: $\mu = \frac{\sum_{i=1}^{n} \mu_i m_i}{\sum_{i=1}^{n} m_i}$

2：计算全局平方的均值: $\nu = \frac{\sum_{i=1}^{n} \nu_i m_i}{\sum_{i=1}^{n} m_i}$

最后是BN过程

算法三:改进后 BN 在第 i 块 GPU 卡上的向前传播算法：
第 i 块 GPU 卡上一个 sub batch 的**输入**: $B_i = \{x_{i,j}\}$ $(j = 0, 1, 2, \dots m_i)$, μ, ν
参数: γ, β
第 i 块 GPU 卡上的**输出**: $\{y_{i,j} = BN_{\gamma,\beta}(x_{i,j})\}$ $(j = 0, 1, 2, \dots m_i)$

计算过程
1：计算方差: $\sigma^2 = \nu - \mu^2$

2：归一化: $\hat{x}_{i,j} = \frac{x_{i,j} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$ （ε 为固定的极小非 0 值，防止除 0 情况发生）

3：计算偏移量: $y_{i,j} = \gamma \hat{x}_{i,j} + \beta$

经过以上三步之后，我们可以得到利用全局信息的均值和方差来做归一化后的值，从而避免了多卡训练中的数据丢失。

（二）反向传播过程的改进

在反向传播中，我们需要把输出数据的梯度作为输入，来计算输入的梯度，并传给下一层神经网络。同时还需要计算 BN 层中参数γ, β的梯度，从而利用梯度下降算法计算新的参数值。传统的算法中，BN 层的反向传播计算过程如下，其中ℓ 是损失函数

传统 BN 在第 i 块 GPU 卡上的反向传播算法（参考文献[1]）：
第 i 块 GPU 卡上一个 sub batch 的**输入**:

$$G_i = \left\{ \frac{\partial \ell}{\partial y_{i,j}} \right\} \ (j = 0, 1, 2, \dots m_i)$$

$$B_i = \{x_{i,j}\} \ (j = 0, 1, 2, \dots m_i)$$

$$\mu_i \ , \sigma_i^2$$

参数: γ, β

第 i 块 GPU 卡上的**输出**: $\left\{ \frac{\partial \ell}{\partial x_{i,j}} \right\} \ (j = 0, 1, 2, \dots m_i), \frac{\partial \ell}{\partial \gamma_i}, \frac{\partial \ell}{\partial \beta_i}$

计算过程

1：计算归一化后数据的梯度: $\frac{\partial \ell}{\partial \hat{x}_{i,j}} = \frac{\partial \ell}{\partial y_{i,j}} . \gamma$

2：计算方差的梯度: $\frac{\partial \ell}{\partial \sigma_i^2} = \frac{-1}{2} (\sigma_i^2 + \varepsilon)^{-3/2} . \sum_{j=1}^{m_i} \frac{\partial \ell}{\partial \hat{x}_{i,j}} . (x_{i,j} - \mu_i)$

3：计算均值的梯度: $\frac{\partial \ell}{\partial \mu_i} = \sum_{j=1}^{m_i} \frac{\partial \ell}{\partial \hat{x}_{i,j}} . \frac{-1}{\sqrt{\sigma_i^2 + \varepsilon}}$

4：计算输入数据的梯度: $\frac{\partial \ell}{\partial x_{i,j}} = \frac{\partial \ell}{\partial \hat{x}_{i,j}} . \frac{1}{\sqrt{\sigma_i^2 + \varepsilon}} + \frac{\partial \ell}{\partial \sigma_i^2} . \frac{2(x_{i,j} - \mu_i)}{m_i} + \frac{\partial \ell}{\partial \mu_i} . \frac{1}{m_i}$

5：计算γ的梯度: $\frac{\partial \ell}{\partial \gamma_i} = \sum_{j=1}^{m_i} \frac{\partial \ell}{\partial y_{i,j}} . \hat{x}_{i,j}$

6：计算β的梯度: $\frac{\partial \ell}{\partial \beta_i} = \sum_{j=1}^{m_i} \frac{\partial \ell}{\partial y_{i,j}}$

我们改进后的技术方案与向前传播时相同，依然是将向后传播的BN分成了预处理(pre-process)，归约(reduce)和归一化(BN)三个步骤，其过程如下：

Backward pre-process

算法四: Pre-process 在第 i 块 GPU 卡上的反向传播算法：
第 i 块 GPU 卡上一个 sub batch 的**输入**:

$$G_i = \left\{ \frac{\partial \ell}{\partial y_{i,j}} \right\} \ (j = 0, 1, 2, \dots m_i)$$

$$B_i = \{x_{i,j}\} \ (j = 0, 1, 2, \dots m_i)$$

第 i 块 GPU 卡上的**输出**: $\varphi_i \ , \phi_i$

计算过程

1：计算梯度的均值: $\varphi_i = \frac{1}{m_i} \sum_{j=1}^{m_i} \frac{\partial \ell}{\partial y_{i,j}}$

2：计算梯度与输入数据乘积的均值: $\phi_i = \frac{1}{m_i} \sum_{j=1}^{m_i} \frac{\partial \ell}{\partial y_{i,j}} . x_{i,j}$

Backward 全局reduce

算法五:全局 reduce 的反向传播算法:

输入: $\varphi_i, \phi_i, m_i\ (i = 0, 1, 2, \dots n)$

输出: φ, ϕ

计算过程

1: 计算全局均值: $\varphi = \frac{\sum_{i=1}^{n} \varphi_i m_i}{\sum_{i=1}^{n} m_i}$

2: 计算全局平方的均值: $\phi = \frac{\sum_{i=1}^{n} \phi_i m_i}{\sum_{i=1}^{n} m_i}$

Backward BN

算法六:改进后 BN 在第 i 块 GPU 卡上的反向传播算法:

第 i 块 GPU 卡上一个 sub batch 的**输入**:

$$G_i = \left\{ \frac{\partial \ell}{\partial y_{i,j}} \right\}\ (j = 0, 1, 2, \dots m_i)$$

$$B_i = \{x_{i,j}\}\ (j = 0, 1, 2, \dots m_i)$$

$$\varphi, \phi, \mu, \sigma^2$$

参数: γ, β

第 i 块 GPU 卡上的**输出**: $\left\{ \frac{\partial \ell}{\partial x_{i,j}} \right\}\ (j = 0, 1, 2, \dots m_i), \frac{\partial \ell}{\partial \gamma_i}, \frac{\partial \ell}{\partial \beta_i}$

计算过程

1: 计算归一化后数据的梯度: $\frac{\partial \ell}{\partial \hat{x}_{i,j}} = \frac{\partial \ell}{\partial y_{i,j}} . \gamma$

2: 计算方差的梯度均值: $\frac{\partial \ell}{\partial \sigma^2}' = \frac{-1}{2} (\sigma^2 + \varepsilon)^{-\frac{3}{2}} . (\phi - \mu\varphi) . \gamma$

3: 计算均值的梯度均值: $\frac{\partial \ell}{\partial \mu}' = \frac{-1}{\sqrt{\sigma^2 + \varepsilon}} . \phi . \gamma$

4: 计算输入数据的梯度: $\frac{\partial \ell}{\partial x_{i,j}} = \frac{\partial \ell}{\partial \hat{x}_{i,j}} . \frac{1}{\sqrt{\sigma^2 + \varepsilon}} + \frac{\partial \ell}{\partial \sigma^2}' . 2(x_{i,j} - \mu) + \frac{\partial \ell}{\partial \mu}'$

5: 计算γ的梯度: $\frac{\partial \ell}{\partial \gamma_i} = \frac{\phi - \mu\varphi}{\sqrt{\sigma^2 + \varepsilon}} . m_i$

6: 计算β的梯度: $\frac{\partial \ell}{\partial \beta_i} = \varphi . m_i$

经过以上三步之后，我们可以得到与单GPU训练计算出的梯度完全一致的结果。

2.3举具体的实例来详述本发明的技术方案（*请举具体的实例详述本发明或者实用新型的技术方案，如方法发明中每个步骤的具体实现方式，实用型新中产品的组成采用的是什么器件等等。*）

我们将该技术方案整合到了深度学习训练框架mxnet中，实现了完整的可执行程序，

并实际运行测试了效果。Mxnet的系统设计分为C++层和python层。C++层主要负责任务调度，内存优化，计算图优化等系统级功能，python层主要是封装完整的训练过程，并提供与用户交互的接口。传统的python层训练过程如下：

```
Mxnet 的传统计算算法：
For each data batch:
    Disperse data batch to each GPU（sub batch）
    For each GPU:
        Forward()
        Backward()
    Update(parameters)
```

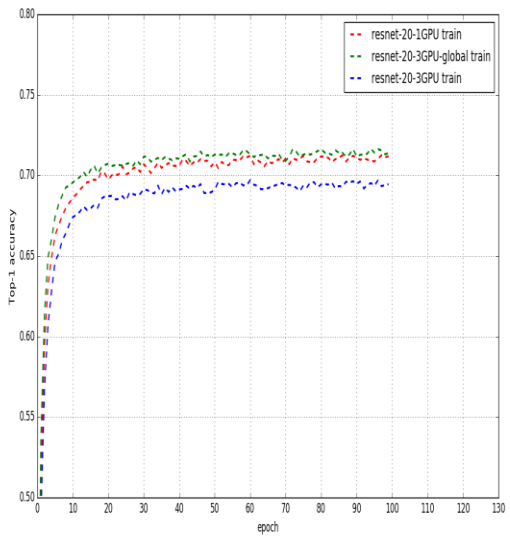
我们的系统修改在这两层都会有所涉及，修改后的系统可以向原来的系统一样正常调用python接口，用户写的程序不需要做任何修改即可正常执行。具体实现步骤如下：

1. 从C++层的计算图中，找出做BN的节点位置，放入BN_forward_location_node和BN_backward_location_node并传给python层。
2. 修改python层的执行模式，让其支持计算图从任意start节点执行到end节点的计算功能,而不是现在的forward()只能做整个计算图从头到尾的计算。
3. 修改BN层的算法，前向传播修改为算法三，反向传播修改为算法六。
4. 前向传播进入BN层之前，添加preprocess_forward层执行算法一；在反向传播进入BN层之前，添加preprocess_backward层执行算法四。
5. 增加新的全局通信节点。在前向传播的preprocess_forward层与BN层之间，添加reduce_forward，执行算法二，在反向传播的preprocess_backward层与BN层之间，添加reduce_backward，执行算法五。

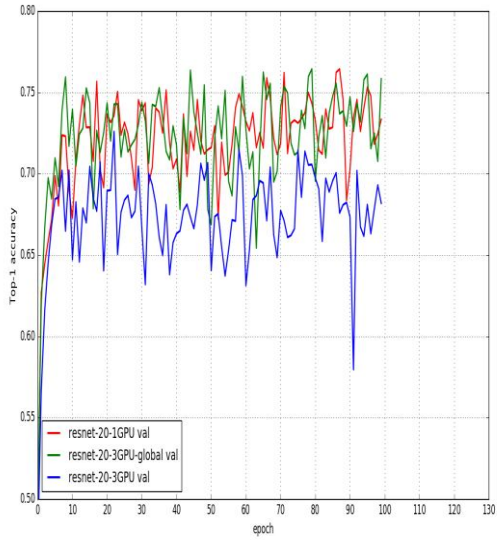
修改后的python层训练过程如下：

```
Mxnet 修改后的计算算法：
For each data batch:
    Disperse data batch to each GPU（sub batch）
    Start=0
    For end in BN_forward_location_node:
        For each GPU:
            Forward(start, end)
            Preprocess_forward()
            start=end
        Reduce_forward()
    For end in BN_backward_location_node:
        For each GPU:
            Backward(start, end)
            Preprocess_backward()
            start=end
        Reduce_backward()
    Update(parameters)
```

2.4 本发明或者实用新型技术方案带来的有益效果*（请通过对技术方案的具体分析得出有益效果，如：由于采用了...使得...从而带来了...的有益效果等）*



图四



图五

图四(训练精度)和图五(检验精度)显示了用我们的方案所达到的训练精度与单 GPU 训练和传统数据并行下多 GPU 训练精度比较。从图中可以看出，我们的方法训练出的模型可以达到与单 GPU 训练相同的精度。与原始的数据并行训练方案相比，精度可以提升15%左右。

该技术方案主要运用在深度学习中对图片分类和分割的模型训练中。这类模型的特点是，单个数据较大(一般一副图片大小为 2M 以上)，而在训练中还要存大量的中间层数据，导致较大的 batch size 会占满整个 GPU 的显存(比如我们在图片分割的模型训练中，一块 GPU 卡最多放 3 张图片就会把显存占满)。假设我们使用 8 张 GPU 卡的机器做数据并行训练，如果不做全局 BN，训练精度只能达到与单卡训练 batch size=3 时相同的精度(70%)；而做全局 BN 之后，其精度可以提升为与单卡训练 batch size=24 时相同的精度(77.2%)。

该技术方案主要运用在单个输入数据较大的场景。因为单个数据较大，较大的 batch 会导致 GPU 上显存占满，从而无法通过继续增大 batch 来提高并行性。此时只有在限定的 batch 大小下，做全局的 BN 来提高训练精度。

3、针对本发明或实用新型要解决的技术问题，若有替代方案则请提供替代方案

本技术方案主要针对深度神经网路的数据并行训练中，批量归一化(batch normalization)层进行修改。主要创新点有：

1 修改了传统BN层的算法，在前向与反向传播算法中，增加了可用于多GPU通信的变量,使整个计算架构便于多卡通信。

2 在前向与反向传播算法中，增加了进入BN层之前的预处理(pre-process)算法和全局归约(reduce)算法。

附件：

参考文献（如专利/论文/标准）

[1] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. CoRR, 2015.