基于数据并行的神经语言模型多卡训练分析

李垠桥，阿敏巴雅尔，薄乐，肖桐，朱靖波，张俐

（东北大学自然语言处理实验室，辽宁沈阳 110819）

摘要：数据并行训练神经语言模型旨在不改变网络结构的同时，大幅度降低训练所带来的时间消耗。但由于多设备之间频繁的数据传输，使得整体加速效果并不理想。本文通过实验对比 All-Reduce 算法和基于采样的梯度更新策略在数据传输上的加速效果，使用了 4 块 NVIDIA TITAN X (Pascal) GPU 设备在循环神经语言模型上进行训练，两种方法分别可获得约 25%和 41%的速度提升。同时，本文还针对数据并行方法的适用性以及不同的硬件设备连接方式对传输速度的影响进行了讨论。

关键词：数据并行；神经语言模型；All-Reduce；采样

中图分类号：TP391　　**文献标识码：**A

Analysis of Data Parallel Methods in Training Neural Language Models via Multiple GPUs

LI Yinqiao, HAN Ambyer, BO Le, XIAO Tong, ZHU Jingbo, ZHANG Li

(NLP Laboratory, Northeastern University, Shenyang, Liaoning 110819, China)

Abstract: Data parallelism aims at reducing time consumption without changing network structure while training neural language model. However, the result is not satisfactory due to frequent data transmission between multiple devices. In this paper, we compare the effect of gradient update strategies based on the All-Reduce algorithm and the sampling-based approach in data transmission. On four NVIDIA TITAN X (Pascal) GPUs, it achieves a speedup of 25% and 41% in recurrent neural language model respectively. Additionally, we discuss the applicability of data parallelism and influence of hardware connection mode.

Key words: data parallelism; neural language model; All-Reduce; sampling

1 引言

使用神经网络进行语言建模依赖于大规模的语料数据，同时更大规模的参数设置一般来说也会对神经语言模型的训练有着正向的作用[1-3]。但当面对海量的数据和大规模的网络参数时，如何更加快速地进行模型训练便成了一个亟待解决的问题。

针对此问题，研究人员引入了 GPU 加快矩阵运算，为了进一步获得速度提升，训练也开始从单一设备转变到多设备并行。其主要方法有两种，数据并行和模型并行[4]。本文主要针对数据并行进行研究，该方法将数据分成若干部分在多个设备上进行训练以达到加速的效果。但该方法的简单实现并未达到令人满意的速度提升[5]，问题在于训练过程中,设备间的数据传输占用大量时间。实验中，我们使用 4 张 NVIDIA TITAN X (Pascal)对循环神经网络进行训练，数据传输的时间占比高达 70%。可以看出减小这部分耗时成为解决多设备训练中的重要问题。

科研人员针对如何在单位时间内传输大量的数据进行了研究，提出了许多可行的方法，如异步参数更新[6]、基于采样的更新[7]等。本文主要针对使用 All-Reduce 算法以及采样策略的神经网络梯度更新进行实验，在不同设备数量下训练前馈神经网络和循环神经网络语言模型[8]，对比分析时间消耗随设备数量的变化趋势。实验中，使用上述两种方法训练的循环神经语言模型相对点对点结构在 4 张 NVIDIA TITAN X (Pascal) 环境下分别可节约 25%和 41%左右的时间。

2 面向神经语言模型的数据传输

2.1 数据并行训练

数据并行的方法最早由 Jeffrey Dean 等

基金项目：本课题得到国家自然科学基金面上项目(No. 61672138)、国家自然科学基金重点项目(No. 61432013)、中央高校基本科研业务费支持。

通讯作者：肖桐（xiaotong@mail.neu.edu.cn）

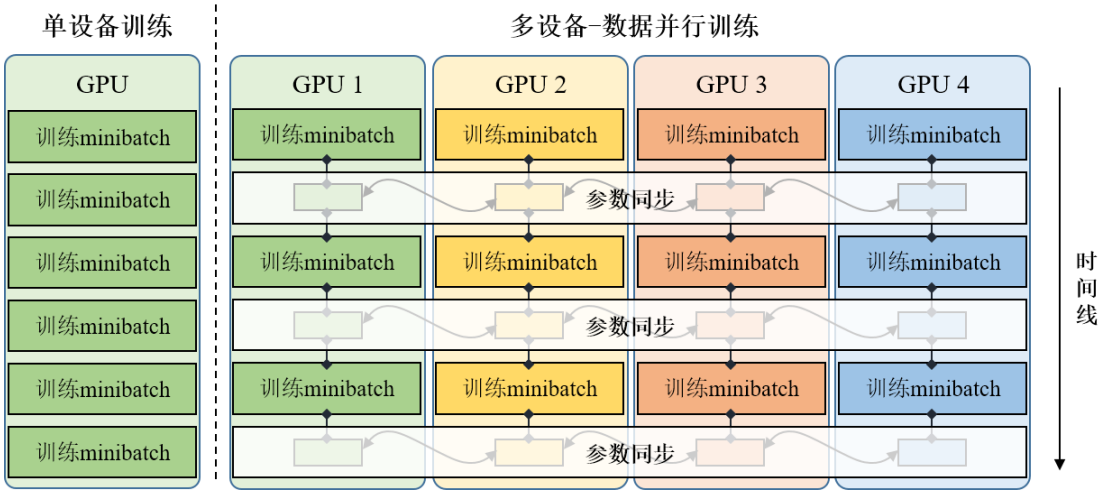

图 1 数据并行训练与单一设备训练过程对比

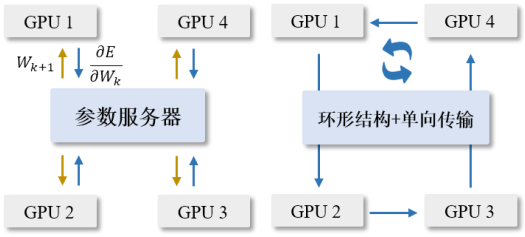

图 2 基于点对点、All-Reduce 的梯度更新结构

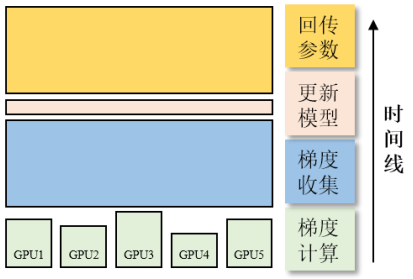

图 3 基于点对点的梯度更新计算时序

人提出[4]，将数据分散到不同的设备中训练，过程如图 1 所示，参数按照下式进行更新。

$$W_{k+1} = W_k - \eta * \sum_{d=1}^{n} (\frac{\partial E}{\partial W_k})_d \qquad (1)$$

式中W_k和W_{k+1}分别是更新前后网络参数，η是学习率，$(\frac{\partial E}{\partial W_k})_d$为 d 设备上的参数梯度。

按照参数同步的方式和频率我们可以对数据并行进行分类。前者主要有点到点传输、基于 All-Reduce 的传输以及基于采样的传输[10]，后两种方式分别从传输速度和传输量上对点对点的方式进行优化，获得了速度上的提升。按照同步频率来划分，数据并行又可分为同步和异步[15]两种，同步的方法要求在收集到系统中全部设备梯度后才进行参数更新，而异步的方法放弃了这一限制，对速度上有一定的提升，但与此同时却带来了梯度过期的问题，一定程度上影响系统的收敛速度，针对该问题通常采用过期值对学习率进行缩放来缓解[12-15]。此外对于支持传输和计算同时执行的设备，双缓冲等实现方法也可对系统训练速度进行提升[16]。

2.2 数据传输方法

2.2.1 点对点的数据传输

点对点数据传输是一种常见的参数同步策略，中心化结构，如图2左侧部分所示。网络中设置一个参数服务器，保存全局的网络参数，其余每台设备在计算好自己的权重梯度后将其发送给参数服务器，然后自身进入等待状态。参数服务器端收集到全部设备的梯度后，将它们累加到自身的网络参数上，之后再将这个更新后的参数值返回给每个计算设备，完成一个 minibatch 的更新。整个过程如图 3 所示，主要分为梯度计算、梯度收集、更新模型和回传参数四个部分。

由于计算机总线带宽有限，因此和数据传输相关的操作会花费较多时间。假设网络中计算梯度的设备数量为n，每个设备上所需传输的数据量大小是K，网络中数据传输的总线带宽是 B，那么在梯度收集和回传参数过程中产生的时间消耗为

算法 1： 使用 All-Reduce 进行梯度累加

输入： 设备数量 n（>1），梯度向量 grad

```
blocks = split(grad, n)
for i = n → 2 do
    send(next, block[(i+devID]%n])
    recv_block = recv()
    block[(i+devID-1)%n]+=recv_block
end for
for i = n → 2 do
    send(next, block[(i+devID+1)%n])
    recv_block = recv()
    block[(i+devID)%n]=recv_block
end for
```

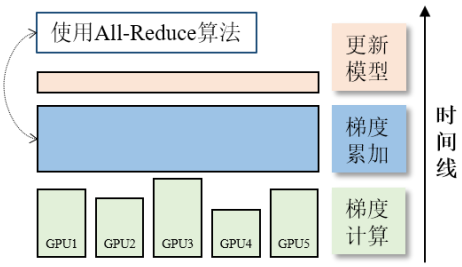

图 4 基于 All-Reduce 的梯度更新计算时序

$$t = 2n * \frac{K}{B} \tag{2}$$

我们可以看出随着设备数量n的增加，传输的时间消耗也随之线性增长，这导致很难通过简单地增加设备数量获得线性增长的运算速度，难以在更多设备上对网络进行训练。

2.2.2 基于 All-Reduce 的数据传输

为减轻参数服务器在数据传输过程中的压力，Linnan Wang 等人提出使用 All-Reduce 算法进行梯度的传递[10]，结构如图 2 右侧部分所示。

整个网络以环状结构进行传输，起初每个设备节点将自身的梯度数组分成 *n* 份，*n* 为环形结构中设备节点的数量。在第一阶段，设备节点将依次发送每一块数据到其下一顺位的设备，并从上一个节点的接收一个块，累积进本设备的对应位置。执行 *n-1* 次后，每个设备中将拥有一个累加了全部设备上对应位次数据的块，之后进入第二阶段。每个设备依次发送自身拥有最终结果的数据块到下一位序的设备，同时根据接收到的数据块进行覆盖更新，同样该步骤按序执行 *n-1* 次后，网络中每个设备均将获得最终的累加结果。算法流程如算法 1 所示。

使用 All-Reduce 算法进行梯度累加，取代了原有方法中梯度收集和回传参数的过程,如图 4 所示，最终完成整个过程的时间为

$$t = 2 * \left(1 - \frac{1}{n}\right) * \frac{K}{B} \tag{3}$$

从公式中我们可以看出，虽然传输时间仍随设备数量 *n* 的增大而增大，但不同于点对点的数据传递，传输时间和设备数量之间不存在线性相关的问题，这一点为引入更多设备参与训练提供了良好的基础。此外，由于该方法并未对模型训练方式及内容进行修改，因此不会对模型本身的性能产生影响。

2.2.3 基于采样的数据传输

不同于 All-Reduce 算法从增大传输速度的角度减小训练耗时，采样的方法希望通过减小数据传输量达到加速的目的[7]。该方法在收集梯度的过程中，从完整的梯度矩阵中抽取出对提高神经语言模型性能更有帮助的部分进行传输，减少了传输的数据量，从而节约了在该过程中的时间消耗。

该方法针对不同层提出了不同的采样方法。如对于输出层权重（大小为 *v*h*，*v* 为词表大小，*h* 为隐藏层节点数目），其中每一行对应着词表中的一个词。为使网络能够更快收敛，在采样过程中希望能够尽可能频繁更新那些经常出现的词，具体的选取策略为

$$V_{all} = V_{base} \cup V_{\alpha} \cup V_{\beta} \tag{4}$$

其中，V_{base}为在当前 minibatch 中出现的词，V_{α}为从词汇表中选择频繁出现的若干词，V_{β}为从词汇表中随机抽取的词以保证系统具有良好的鲁棒性，在测试集上更稳定。同样在输入层和隐藏层也有不同的采样策略。另外，由于本方法更加频繁地对能够加快收敛的梯度部分进行更新，使得模型收敛速度加快，与此同时模型本身性能变化不大。

3 对比

3.1 实验系统

本文相关实验使用东北大学自然语言处理实验室 NiuLearning 深度学习平台，结

GPU 数量		1	2	3	4
时间消耗(s)	baseline	3171.9	2499.2	2027.5	1717.7
	baseline+AllReduce	3089.7	2188.3	1521.9	1273.4
节约时间(s)		82.164	310.82	505.66	444.28
加速占比(%)		2.5903	12.437	24.939	25.864

表 1 基于 All-Reduce 梯度更新策略在循环神经语言模型中每个 Epoch 的时间消耗情况

GPU 数量		1	2	3	4
时间消耗(s)	baseline	3171.9	2499.2	2027.5	1717.7
	baseline+sampling	3015.7	1868.5	1287.4	1003.7
节约时间(s)		156.20	630.63	740.15	714.02
加速占比(%)		4.9245	25.233	36.504	41.567

表 2 基于采样的梯度更新策略在循环神经语言模型中每个 Epoch 的时间消耗情况

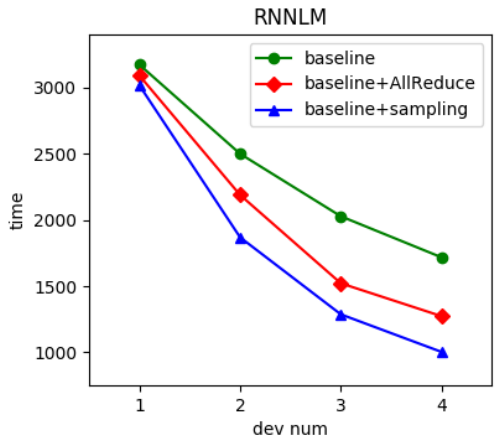

图 5 基于 All-Reduce 和采样策略的时间消耗对比

合 NCCL 开源框架，在 4 块 NVIDIA TITAN X (Pascal)设备上进行。主要对比不同设备数量下，基于 All-Reduce 算法和采样的更新策略对循环神经语言模型训练速度的影响。此外本文也在前馈神经语言模型[9]中使用 All-Reduce 算法进行实验，分析数据并行在不同网络结构下的适用性问题。

本文实验数据使用 Brown 英文语料库（1,161,169 词，57341 句），从中抽取 40,000 句子作为训练集，统计 49,036 词作为语言模型词汇表。在模型参数方面，循环神经网络和前馈神经语言模型输入层和隐藏层节点个数均为 1024，minibatch 为 64。前馈神经网络训练 5-gram 语言模型。

3.2 实验结果及分析

3.2.1 加速效果及变化趋势

实验中将基于点对点进行数据传输的系统作为基线，在 4 张 GPU 设备上获得约 25%的加速效果，具体实验结果见表 1。使用基于采样的梯度更新策略在同样参数设置下，4 张卡加速比达 41%左右，具体实验结果如表 2 所示，整体趋势如图 5 所示。

在基线系统的时间消耗中，由于梯度的计算和更新模型不存在内存和显存的交互，因此速度较快。而对于梯度的收集和参数回传，其占用的时间随设备数量的增多而线性变大，因此每个 minibatch 所消耗的时间为

$$t = 2n * \frac{K}{B} + t_{cal\&update} \qquad (5)$$

其中$t_{cal\&update}$为梯度计算和模型更新所消耗的总时间。同理，根据图4我们可以看出，使用了 All-Reduce 算法的梯度累加替代了原本方案中的梯度收集和参数回传的过程，同时这部分的时间消耗并不随设备数量线性增长。在使用了该算法进行梯度更新的情况下，每个 minibatch 所花费的时间为

$$t = 2 * \left(1 - \frac{1}{n}\right) * \frac{K}{B} + t_{cal\&update} \qquad (6)$$

在整个的训练过程中，由于样本数量是固定的，因此 minibatch 数量也是不变的。而多设备训练将样本分散到不同设备上并行计算，使得在同一时刻整个系统见到的 minibatch 个数随设备数量线性增加，换句话说总时间消耗将降为原本的$1/n$，因此在每一轮训练过程中，基线和基于 All-Reduce 的方法在时间消耗上分别为

$$t_{baseline} = \left(\frac{2K}{B} + \frac{t_{cal\&update}}{n}\right) * bn \qquad (7)$$

设备数量	baseline	baseline+All-Reduce	节约时间
1	$\left(t_{gather\&back} + t_{cal\&update}\right) * bn$	$t_{cal\&update} * bn$	$t_{gather\&back} * bn$
2	$\left(\frac{2K}{B} + \frac{t_{cal\&update}}{2}\right) * bn$	$(\frac{2K}{B} * 0.25 + \frac{t_{cal\&update}}{2}) * bn$	$(\frac{2K}{B} * 0.75) * bn$
3	$\left(\frac{2K}{B} + \frac{t_{cal\&update}}{3}\right) * bn$	$(\frac{2K}{B} * 0.22 + \frac{t_{cal\&update}}{3}) * bn$	$(\frac{2K}{B} * 0.78) * bn$
4	$\left(\frac{2K}{B} + \frac{t_{cal\&update}}{4}\right) * bn$	$(\frac{2K}{B} * 0.19 + \frac{t_{cal\&update}}{4}) * bn$	$(\frac{2K}{B} * 0.81) * bn$

表 3 在 1-4 个设备上执行一轮训练所需理论时间

GPU 数量		1	2	3	4
时间消耗(s)	baseline	317.22	1280.4	1242.6	1228.8
	baseline+AllReduce	216.07	1245.4	1095.4	903.58
节约时间(s)		101.15	35.003	147.18	325.23
节约占比(%)		31.886	2.7337	11.845	26.467

表 4 基于 All-Reduce 梯度更新策略在前馈神经语言模型中每个 Epoch 的时间消耗情况

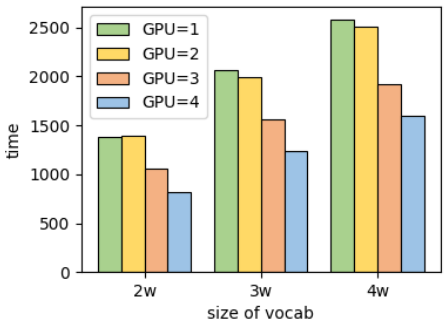

图 6 不同词表大小下基于 All-Reduce 的时间消耗

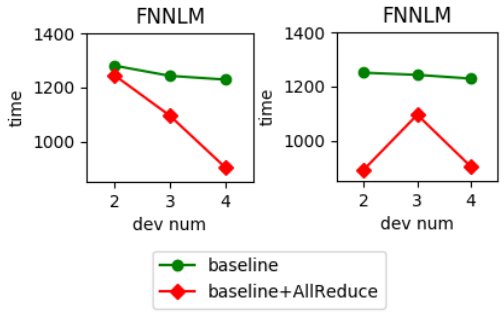

图 7 不同设备配置下基于 All-Reduce 的时间消耗

$$t_{AllReduce} = \left[\frac{2K}{B} * \left(\frac{1}{n} - \frac{1}{n^2}\right) + \frac{t_{cal\&update}}{n}\right] * bn \quad (8)$$

其中bn为整个训练集中 minibatch 的数量。根据上述公式，我们可以得到理论上在 1-4 张卡的状态下每轮执行所需要的时间，如表 3 所示。其中当设备数量是 1，不使用 All-Reduce 的情况下，由于无需通过总线进行数据传递，将这部分耗时用$t_{gather\&back}$表示。从表 3 中我们可以看出，随着设备数量的增长，节约的时间越来越多，这一点从图 5 的实验结果中也能观察到。

从表 2 中实验结果我们可以看出，使用采样的梯度更新策略可以获得更高的加速比。不同于 All-Reduce 试图找到一种更快速的传输策略，该方法通过减少数据传输量来降低多设备训练中大量的时间消耗。由于二者出发点不同，因此理论上可以同时使用两种方法对训练过程进行加速。

3.2.2 数据并行算法适用性

我们使用不同大小的词汇表，通过 All-Reduce 算法训练循环神经语言模型，时间消耗如图 6 所示。可以看到随着词表的减小，多设备训练的时间优势逐渐降低。

此外我们在前馈神经语言模型上进行实验，参数相关设置与循环神经网络相同，具体实验结果见表 4。在多设备情况下，随着设备数量的增多，前馈神经网络在使用 All-Reduce 算法前后变化趋势如图 7 左侧部分所示，类似于循环神经网络。但从表 4 中我们可以看出，当设备数为 1 时，训练 1 轮所需时间远小于多设备训练的方法。以上两

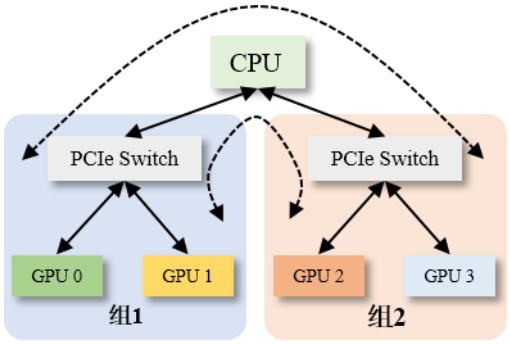

图 8 实验平台拓扑结构

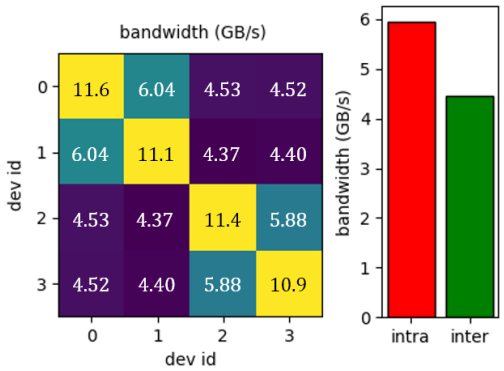

图 9 不同设备之间数据传输带宽

组实验均出现了单设备训练优于多设备并行的现象。循环神经语言模型实验中，由于词表大小的减小，梯度计算耗时变低，导致并行优势被数据传输所掩盖。同理前馈神经网络相较循环神经网络在结构上更加简单，数据传输占据了耗时的主要部分，即使使用 All-Reduce 算法对传输时间进行稀释，也无法抵消其所带来的耗时问题。从表 3 的理论推导中我们也可以看出，当使用多设备（$n>1$）训练神经网络的时候，时间消耗均与 K 有着很强的线性关联，当所需传输的数据量较大时，多设备训练的实际效果并不会很好。

3.2.3 硬件连接方式对速度的影响

在实验过程中发现，设备之间硬件的连接方式也会对传输速度产生影响。在前馈神经网络语言模型的实验中，当设备数量为 2 且工作节点使用 0 号和 1 号 GPU 进行训练时，时间消耗曲线如图 7 中右侧部分所示，左侧部分为使用 0 号和 2 号设备所得，我们可以看出，当选择不同设备进行数据传输时，其所消耗的时间也有可能不同。我们实验平台上的硬件拓扑结构如图 8 所示，0 号和 1 号卡为一组，2 号和 3 号为一组，均使用 PCIe Switch 进行连接。

由于组内进行数据传输无需通过 CPU，占用总线带宽，因此传输速率较快，组间由于需要经过多个设备对数据进行转发，因此速度相对慢一些，1-4 卡之间互相传输数据带宽如图 9 左侧部分所示，其中颜色越浅带宽越高，传输速率越快，图 9 右侧给出组内平均带宽为 5.96GB/s，组间 4.46GB/s。

由于 All-Reduce 算法采用环形结构让设备之间进行数据传输，因此只要系统中存在任意两台设备间传输速度慢，那么就会拖慢整体系统的传输速率。因此在前述两张卡的实验中，为保证实验公平性，我们使用 0 号和 2 号进行实验，保证多设备的每组实验中均存在组间的数据传递。同理，在基于采样的数据传输过程中，其数据传输方式仍为点到点的传输，慢速的连接只会影响节点本身与参数服务器之间的传输过程，对整体的时间消耗影响有限。

4 总结

本文主要针对如何降低多设备训练神经语言模型中的时间消耗进行实验，对比不同梯度更新策略的加速效果。

- **加速效果：**使用 All-Reduce 的梯度收集策略在循环神经语言模型上可获得 25%左右的加速效果，同时随着设备数量的增加加速效果更加显著；使用基于采样的梯度更新方法可达到约 41%的加速。
- **数据并行适用性：**使用多设备训练神经网络在不同模型下速度变化趋势稍有不同。实验发现对于大量数据传输占主导的网络模型，多设备训练反而会降低整体运行速度。在这种情况下即使使用了 All-Reduce 等的加速方法也很难获得与单设备可比的运行速度。而对于数据传输量偏小的模型，在实际使用中比较适合在多设备上并行训练。
- **硬件连接方式对速度的影响：**实验中发现，设备之间不同的硬件连接方式会对传输速度产生影响。针对该特性，我们在未来工作中考虑将采样算法与 All-Reduce 进行结合，对于传输较慢的设备在采样中给予相对较低的采样率，实现能者多劳，不让某些设备的传输速度拖慢整体。

参考文献

[1] Jozefowicz R, Vinyals O, Schuster M, et al. Exploring the Limits of Language Modeling[J]. 2016.

[2] Hannun A, Case C, Casper J, et al. Deep Speech: Scaling up end-to-end speech recognition[J]. Computer Science, 2014.

[3] Amodei D, Anubhai R, Battenberg E, et al. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin[J]. Computer Science, 2015.

[4] Dean J, Corrado G S, Monga R, et al. Large scale distributed deep networks[C]. International Conference on Neural Information Processing Systems. Curran Associates Inc. 2012:1223-1231.

[5] Bengio Y, Senecal J S. Adaptive importance sampling to accelerate training of a neural probabilistic language model.[J]. IEEE Transactions on Neural Networks, 2008, 19(4):713-22.

[6] Zhang S, Zhang C, You Z, et al. Asynchronous stochastic gradient descent for DNN training[C]. IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013:6660-6663.

[7] Tong Xiao, Jingbo Zhu, Chunliang Zhang, et al. Fast Parallel Training of Neural Language Models. IJCAI, 2017.

[8] Mikolov T, Karafiát M, Burget L, et al. Recurrent neural network based language model[C]. INTERSPEECH 2010, Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September. DBLP, 2010:1045-1048.

[9] Bengio Y, Vincent P, Janvin C. A neural probabilistic language model[J]. Journal of Machine Learning Research, 2003, 3(6):1137-1155.

[10] Wang L, Wu W, Bosilca G, et al. Efficient Communications in Training Large Scale Neural Networks[J]. 2017.

[11] Zhang S, Lecun Y, Lecun Y. Deep learning with elastic averaging SGD[C]. International Conference on Neural Information Processing Systems. MIT Press, 2015:685-693.

[12] Wei Zhang, Suyog Gupta, Xiangru Lian, et al. Staleness-aware async-sgd for distributed deep learning. IJCAI, 2016.

[13] Ho Q, Cipar J, Cui H, et al. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server[J]. Advances in Neural Information Processing Systems, 2013, 2013(2013):1223.

[14] Suyog Gupta, Wei Zhang, and Josh Milthrope. Model accuracy and runtime tradeoff in distributed deep learning. arXiv preprint arXiv:1509.04210, 2015.

[15] Augustus Odena. Faster asynchronous sgd. arXiv preprint arXiv:1601.04033, 2016.

[16] Seide F, Fu H, Droppo J, et al. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs[J]. Interspeech, 2014.