Data optimization for large batch distributed training of deep neural networks

Shubhankar Gahlot
Oak Ridge National Lab
Oak Ridge, USA
Email: sgahlot@hawk.iit.edu

Junqi Yin
Oak Ridge National Lab
Oak Ridge, USA
Email: yinj@ornl.gov

Mallikarjun (Arjun) Shankar
Oak Ridge National Lab
Oak Ridge, USA
Email: shankarm@ornl.gov

Abstract—Distributed training in deep learning (DL) is common practice as data and models grow. The current practice for distributed training of deep neural networks faces the challenges of communication bottlenecks when operating at scale, and model accuracy deterioration with an increase in global batch size. Present solutions focus on improving message exchange efficiency as well as implementing techniques to tweak batch sizes and models in the training process. The loss of training accuracy typically happens because the loss function gets trapped in a local minima. We observe that the loss landscape minimization is shaped by both the model and training data and propose a data optimization approach that utilizes machine learning to implicitly smooth out the loss landscape resulting in fewer local minima. Our approach filters out data points which are less important to feature learning, enabling us to speed up the training of models on larger batch sizes to improved accuracy.

I. INTRODUCTION

Distributed strategies for scaling DL have become the cornerstone of solutions to the problems that arise due to ever growing data and model sizes. Scaling a DL workload requires careful calibration of compute, I/O, and communication among the compute nodes [1]. In addition, we need to use large batch sizes to achieve weak scaling to make full use of the parallel hardware. We achieve training speedup by increasing the overall throughput of the system with fewer updates of individual model replicas. Distributed strategies, however, present a set of challenges [2] that arise when using large batches across distributed computing platforms:

Challenge 1: Larger mini-batch sizes average the gradients for large samples thus reducing the variance and providing better estimate of model weights [3]. The goal is to take bigger step sizes and in turn make the optimization algorithm progress faster. However, as shown in [4], increasing the mini-batch size leads to a drop in accuracy and leads to the generalization gap - difference in accuracy between training and testing.

Challenge 2: When using large clusters, it is harder to achieve near-linear scalability as the number of compute nodes increases, especially for models with a high communication bandwidth requirement. In both data parallel and model parallel distributed strategies, the communication step usually becomes the bottleneck as the number of GPUs increases. To improve performance for such distributed training systems, we need faster compute, more efficient bandwidth utilization, and more efficient collective primitives that can handle a system with thousands of GPUs.

It has been shown [2, 5] that the loss landscape is a function of both the DL model and the training data. Current scaling techniques focus more on parallelizing the workloads through data parallel training, but they suffer at large scale as the training batch size grows. This is because the likelihood of the optimization process being stuck in a local minima increases with increasing batch size. Our goal is to empirically identify techniques to effectively smooth the loss landscape and also develop guidelines for optimizing batch sizes for training. We address this by exploring the effect of batch size on model convergence and accuracy. Our experiments apply to ResNet models using the CIFAR-10 [6] data set. Our contributions are as follows:

- We present empirical evidence for the non-linear correlation between batch size and model accuracy for increasing batch sizes from 16 to 20,480. We provide recommendations for the appropriate batch sizes to use for the particular ResNet models.
- We develop techniques using machine learning (ML) to empirically characterize data quality, and propose a data optimization approach to model scaling. We cluster data points using ML and remove those classified as noise. We use the remaining data to train another network (while keeping the testing data unchanged) to equivalent and even higher levels of accuracy for certain large batch sizes thus saving system bandwidth and training time.
- We compare the scaling efficiency for ResNet models between PyTorch's built-in distributed data parallel (DDP) and the Horovod [7] communication library for up to 1536 V100 GPUs and find that the overheads of the dynamic messaging queue in Horovod get in the way of achieving the benefits expected from improved

overlapping of communication and computation.

The rest of the paper is structured as follows. We discuss the background and context for our work in Section II. We present the details of our methodology and experiments in Section III. Section IV discusses our results along with findings and recommendations. We conclude in Section V with a discussion of future extensions.

II. Background and Context for Data-optimized Scaling

A. Background

We list a few concepts necessary for the description that follows.

Dimensionality reduction: Dimensionality reduction is the process of transforming data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension.

Feature extraction using CNN: Feature extraction may be deemed a data reduction technique. In a CNN the output of a convolutional layer captures features (feature maps) of the input data. The last convolutional layer captures the finest features (edges, curves, etc) from the data that it is trained on. Hence, we are interested in the output of that last convolutional layer and this process in general is called feature extraction using a CNN [8].

t-distributed stochastic neighbor embedding (t-SNE): t-SNE is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional data sets [9].

Clustering: Cluster analysis or clustering is an unsupervised machine learning technique in which more similar (based on some metric) data points group together to form a cluster. There are various criteria for clustering. We use DBSCAN [10] for clustering data points in feature space because it is a density based technique and therefore doesn't assume any apriori distribution and can be scaled to large number of data points easily.

B. Data Quality Context and Scaling Objectives

- *Defining data quality:* While there has been some work on relating physical attributes of data sets such as data set equilibrium, size, quality of label and contamination to model quality [11], there has been little to no significant work on how intrinsic properties of a data set affect model training performance. A central reason is because data quality is defined differently for different types of data - ranging widely from domain specific to broad information theoretic measures. For example, quality metrics for traditional image data are different for speech signal data or other types of higher order spectral image data. Our approach is to use machine learning to filter out data points which are not relevant to training effectiveness. This gives us a way of thinking about how the quality of data elements affect the training outcomes. We use unsupervised techniques like dimensionality reduction

and clustering to categorize data points and filter out those we classify as "noise" in feature space. Various dimensionality reduction, clustering, and data sampling techniques may be used for this step with different degrees of effectiveness.

- *Feature (space) noise vs. Gradient descent noise:* Feature (space) noise refers to the kinks and troughs in the loss landscape as described in [5] that can hinder the optimizer to advance towards the global minima whereas gradient descent noise refers to the noise in stochastic gradient descent (SGD) which gives the gradient descent its stochastic nature. It is known that as the batch size increases, stochasticity decreases, and therefore we can improve generalization either by adding noise to the SGD or by smoothing out the loss landscape. (This is similar to the role of skip connections in ResNets [5].)

- *Smoothing the loss landscape:* The loss landscape is shaped by the model as well as by the data. The higher dimension "topography" of the loss function determines how well the model will converge and to what accuracy. There can exist kinks and twirls in the loss landscape due to data points irrelevant to feature learning, which can make the model convergence a challenge especially at large batch sizes. If we can identify and remove those data points, the loss landscape would become smoother and easier for the model to move towards convergence. By classifying noise in feature space and removing them from the data set, we aim to speed up convergence.

- *Distributed large batch training:* There has been significant work applying data distribution to scale deep learning workloads. Models using batch sizes of 64k have been trained in under 4 mins [12] using techniques such as Layer-wise Adaptive Rate Scaling (LARS) [13] and Adaptive Batch Sizing. However, we focus our attention on scaling from an empirical, intrinsic data quality (or effectiveness) perspective.

In the work we describe here, we take a data optimization approach to model scaling and connect the effectiveness of a data item to the features it enables in a trained model. We remove the data points classified as noise to feature learning and retrain a new model at scale. Our hypothesis is that clustering (and/or sampling) and removing data points classified as noise in feature space will filter data in such a way that it will smooth out the loss landscape. This will facilitate gradient traversal and thereby accelerate model convergence.

III. Methodology and Experiments

A. Conceptual Overview

We use a pretrained network and treat data quality as the output of a machine learning technique which can be used to remove data elements less significant to training. This helps smooth the loss landscape and thus acts as a data-optimized approach to distributed training and model scaling. Figure 1 outlines this process. The pipeline begins at the left by processing the data in a pretrained network. Next,
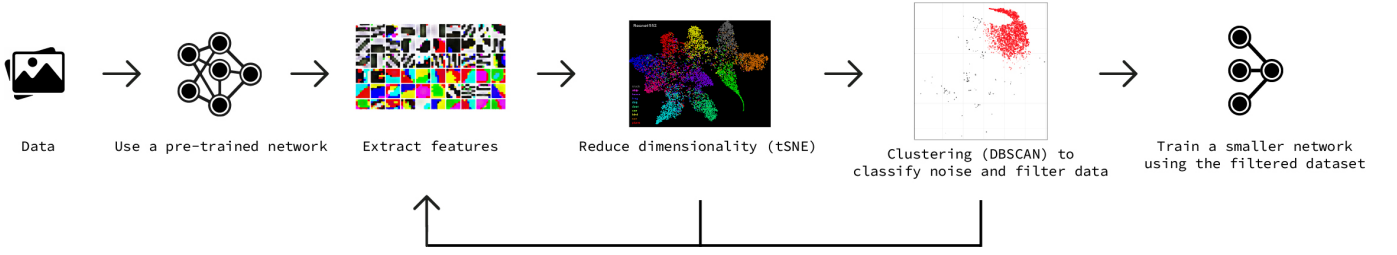
Fig. 1. Steps showing the process of applying data optimized strategy by (1) fine tuning a pretrained network, (2) Extracting and reducing dimensionality by PCA / t-SNE, (3) Clustering using DBSCAN to remove noise and (4) Training another network faster and at improved accuracy.

the data elements that apply to the primary clusters of the penultimate $(n-1\text{th})$ layer are compactly represented through dimensionality reduction (with PCA and t-SNE). These data elements are clustered using DBSCAN to remove what we consider "noise". Finally, we train another smaller network, faster, and at improved accuracy.

More formally, let us represent CIFAR-10 data set by a matrix as follows:

$$\mathbf{X_{50k}} \in \mathbb{R}^{50000\times3\times28\times28} - (1)$$

We represent a batch as follows:

$$\mathbf{X_b} \in \mathbb{R}^{b\times3\times28\times28} - (2)$$

(*NCHW*) where *N or* **b** is the batch size. Now let's define the feature extractor neural network (NN) by

$$\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta}) - (3)$$

where $\mathbf{x} \in \mathbb{R}^{3\times28\times28}$ is one of the image examples and $\boldsymbol{\theta}$ is the value of parameters NN learns that result in the best function approximation. Since **y** is the set of features and output of the penultimate layer therefore,

$$\mathbf{y} \in \begin{cases} \mathbb{R}^{512\times1} & \text{for ResNet-18 and 34,} \\ \mathbb{R}^{2048\times1} & \text{for ResNet-50, 101 and 152} \end{cases} - (4)$$

Now, we further reduce the dimensionality of **y** using t-SNE *t*.

$$\mathbf{y}^{'} = t(\mathbf{y}, \boldsymbol{\phi}) - (5)$$

where t maps $\mathbf{y} \rightarrow \mathbf{y}^{'} \in \mathbb{R}^{2\times1}$ and is parameterized on ϕ. Next, we perform clustering on each class individually using DBSCAN algorithm to classify and filter out noise. Let d denote the DBSCAN function, parameterized on $\boldsymbol{\gamma}$, that maps $\mathbf{y}_{\mathbf{c}}^{'} \rightarrow \mathbf{y}_{\mathbf{c}\alpha}^{'}$ where $c \in [1,2,3,4,5,6,7,8,9,10]$ denotes individual class label and $\alpha \in [0,1]$, 0 for noise and 1 otherwise.

$$\mathbf{y}_{\mathbf{c}\alpha}^{'} = d(\mathbf{y}_{\mathbf{c}}^{'}, \boldsymbol{\gamma}) - (6)$$

Therefore, $\mathbf{y}_{\mathbf{c0}}^{'}$ and $\mathbf{y}_{\mathbf{c1}}^{'}$ denote data points classified as noise and not noise respectively for class **c**. Finally, we combine all images i.e. $\mathbf{y}_{\mathbf{c1}}^{'}s$, resulted from filtering and train a smaller network.
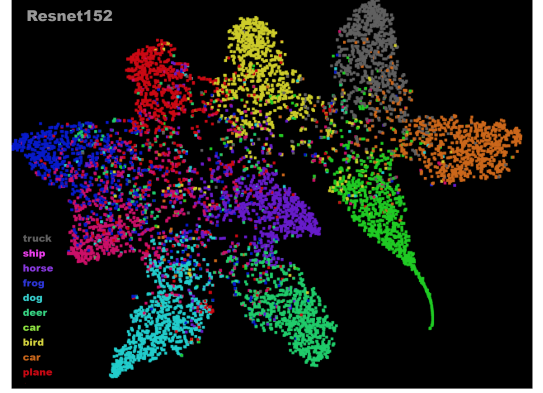

Fig. 2. Visualizing output of extracted features from the fine-tuned ResNet-152 using t-SNE.

B. Process

We expand on the data set details for the steps listed above.

1) Initial training:: First, we train ResNet-152, 101, 50, 34 and 18 from scratch on the full CIFAR-10 data set on a single GPU with a batch size of 32, 64, 128 and 256 as well as perform distributed training on 8 GPUs with batch sizes ranging from 16 to 20,480. We call this complete CIFAR-10 data the *Full* data set. We also use a pretrained model as a point of comparison and find it helps identify features effectively thus saving time and computation.

2) Extract features using CNN: Next, we focus on the last fully connected layer to extract features using the model as seen in Figure 2 [14]. There are 512 neurons in the penultimate layer of ResNet-18 and ResNet-34 models and hence will produce 512 features whereas ResNet-50, 101, and 152 will produce 2048 features.

3) Reduce dimensionality using t-SNE: We further reduce the dimensionality of these features to 2 dimensions using t-SNE (and to also help visualize them.) The t-SNE visualization using ResNet-152 of CIFAR-10 test set for all classes shows how the model has learnt to classify the CIFAR-10 data set effectively. We now use the same model to visualize the training data set and apply the DBSCAN clustering technique to classify noise in the next step.

4) Clustering to classify and filter data: Next, we cluster the training data set using DBSCAN and filter out the data points (images) classified as noise. (These appear as grey dots

TABLE I
AMOUNT OF DATA REDUCTION

Network used for feature extraction	Filtered data size (reduction)
ResNet-152	98.76% or 49380 images (1.24%)
ResNet-101	94.87% or 47438 images (5.12%)
ResNet-50	92.66% or 46333 images (7.34%)
ResNet-34	93.45% or 46725 images (6.55%)
ResNet-18	93.79% or 46895 images (6.21%)

in Fig 3 - source code is available in [14]. Our code includes an interactive visualizer for t-SNE and DBSCAN which can be used to glance over the images classification - for user oversight.) We remove the images classified as noise from the CIFAR-10 data set and re-purpose the remaining images to train a smaller network. We call our new data set *Network filtered* data.
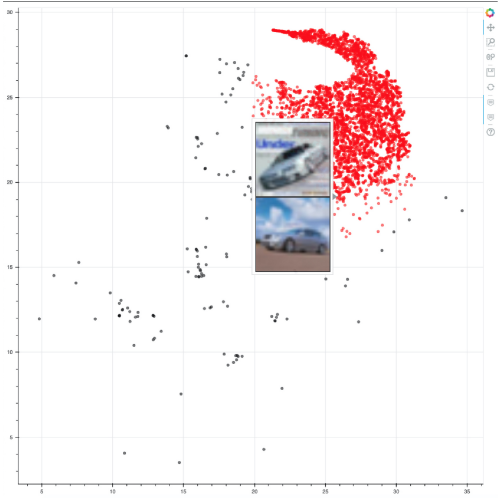

Fig. 3. Applying DBSCAN to the t-SNE output, noise shown in black (Automobile).

Different networks (ResNet-18, 34, 50, 101 or 152) achieve different levels of data reduction as shown in Table I.

5) Train a smaller network: Finally, we use the Network filtered data from ResNet-101 to train ResNet-18. Our rationale in using ResNet-101 is to strike a balance between accuracy and data reduction. Using ResNet-101 to reduce data shows promising results and strikes a balance between data reduction and accuracy.

IV. EMPIRICAL OBSERVATIONS

A. Batch Size Considerations

We train different ResNet-* models on CIFAR-10 data set for 50 epochs using the SGD optimizer and empirically establish for a wide range of batch sizes (from 16 to 20,480) that model accuracy is not linearly correlated with the batch size. Our observations suggest that there is a preferred batch size for this class of models at which the highest accuracy will be achieved at different learning rates.

We start with a set of rounds of single GPU training then move to multi GPU distributed training to verify this
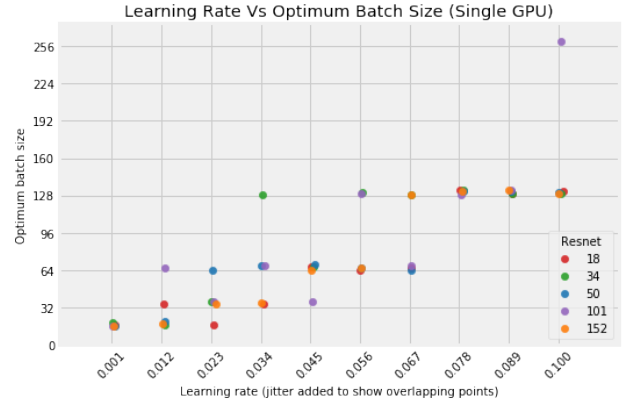

Fig. 4. Learning rate vs. Preferred batch size for single GPU

behavior. This study empirically identifies that there exists an optimal batch size as seen in Figure 4 at which the highest accuracy is achieved and beyond which accuracy drops gradually, as shown in Figure 5, as the batch size increases. This low accuracy for smaller batch sizes applies to batch normalization computation and is explored further in [15]. Briefly, batch normalization (BN) addresses the problem of *internal covariate shift* in the neural network by reducing the dependency of the distribution of the input activations of each layer on all the preceding layers. Let us consider batch X_b from (2) of size of b

$$\mu_{X_b} = \frac{1}{b}\sum_{i=1}^{b} x_i, \quad \text{and} \quad \sigma_{X_b}^2 = \frac{1}{b}\sum_{i=1}^{b}(x_i - \mu_{X_b})^2 - (7)$$

For a layer of the network with d-dimensional input, $x = (x^{(1)}, ..., x^{(d)})$, each dimension of its input is then normalized separately,

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \hat{\mu}_{X_b}^{(k)}}{\sqrt{\hat{\sigma}_{X_b}^{(k)^2} + \epsilon}} - (8)$$

where $k \in [1, d]$ and $i \in [1, m]$; $\mu_{X_b}^{(k)}$ and $\sigma_{X_b}^{(k)^2}$ are the per-dimension mean and variance, respectively. The normalized values $\hat{x}_i^{(k)}$ are then further scaled and shifted by the learned parameters $\gamma^{(k)}$ and $\beta^{(k)}$

$$y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)} - (9)$$

For the case of a convolutional layer, with a feature map of size $p \times q$ and batch size b, the sample size for the estimate of $\hat{\mu}_{X_b}^{(k)}$ and $\hat{\sigma}_{X_b}^{(k)^2}$ is given by $b \times p \times q$, while for a fully-connected layer the sample size is simply equal to b. From (7) it can be concluded that for very small batch size b, the estimation of the batch mean and variance can be very noisy, which may limit the effectiveness of BN in reducing the covariate shift. On the contrary, however, it is well known that too large of a batch size will lead to sub-optimal model convergence and thus poor generalization (although currently it's not known why this is
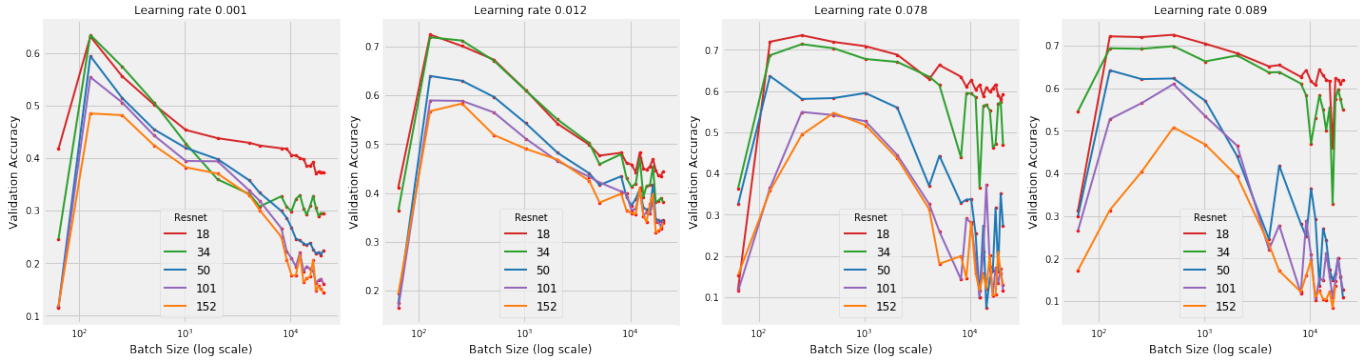
Fig. 5. Large batch size vs. Accuracy for different lr. Accuracy increases until it reaches an optimum batch size and then starts to gradually decrease.

so). For the convex functions we are trying to optimize, there is an inherent tug-of-war between the benefits of smaller and bigger batch sizes.

Batch size vs. Accuracy curves for all ResNet models in Figure 5 tend to run closer and peak together for lower learning rates and diverge as learning rate is increased. This is because for smaller learning rates the optimization step is small for all networks and the resulting variance irrespective of the different loss contours for the networks. Separately, accuracy values for smaller networks like ResNet-18, 34 tend to drop gradually whereas accuracy for bigger networks drops suddenly after achieving the optimal accuracy. This behavior may be attributed to the fact that the larger networks have more complex loss function shapes and hence the optimization step often gets trapped in a local minima.

B. Data Optimized Approach Results

Here we modify the loss landscape by removing noisy data points for ResNet-18 for both small (<256 images) as well as large batch sizes (32 images per GPU $\times$ 8 GPUs i.e. 256 images to 1310 images per GPU $\times$ 8 GPUs i.e. 10480 images). We compare our results for the *Network filtered* data against *Full* CIFAR-10 data set and *Randomly filtered* data. We experimented using multiple runs of different weight decay (L2 reg) values of 0.005, 0.01, 0.012 and 0.0135 for small batch sizes and multiple runs of same weight decay value of 0.01 for large batch sizes.

1) Small batch training: For a batch size of 256, as shown in Figure 6, the network achieves top-1 accuracy of 83.2% at weight decay (L2 reg) of 0.005 on *Network filtered* data as compared to 82.6% at weight decay of 0.012 on *Full* data and 80.9% at weight decay of 0.0135 on *Randomly filtered* data. However, as the batch size is reduced, the accuracy of the *Network filtered* data decreases. For batch size of 64 top-1 accuracy on *Network filtered* data is 81.8% at weight decay of 0.005 as compared to 79.2% for *Randomly filtered* data at weight decay of 0.012 and for batch size of 128 top-1 accuracy on *Network filtered* data is 82.7% at weight decay of 0.005 as compared to 80.5% for *Randomly filtered* data at weight decay of 0.005. Accuracy on *Network filtered* data always remains higher than on *Randomly filtered* data which is to be expected

because the network filtered data is intrinsically biased towards more accurate classification input data elements.

The neural network does better on *Network filtered* data does than both *Randomly filtered* and *Full* data for smaller regularization values. Thus, we use the lowest regularization value of 0.005 for doing large batch size analysis which we shown in Figure 7 and discuss next.

2) Distributed large batch training: Our data optimized strategy to improve accuracy by manipulating the loss landscape shows promising results for batch sizes larger than 6k as seen in Figure 7. For an average of 4 runs and batch sizes of 6272 to 10368, the neural network consistently achieves a higher top-1 accuracy on *Network filtered* data than on *Full* data. For a batch size of 6272, the network achieves top-1 accuracy of 77.5% on *Network filtered* data and 75.2% on *Full* data. When compared to training on *Randomly filtered* data the network consistently achieves a higher accuracy for all batch sizes on *Network filtered* data. However, for smaller than 6k batch sizes the accuracy curve on *Network filtered* data is well within the error bars of the accuracy curve on *Full* data so we can conclude that the model performs as well on the *Network filtered* data as it performs on *Full* data. Also, as expected the variance in accuracy is higher for larger batch sizes because the larger forward optimization step tends to wiggle around the global minima. A likely cause is that the lack of data points removed is being compensated by the transformations applied to the remaining data points.

C. Communication Characteristics and Performance

Besides the data-optimized reduction resulting in improved convergence discussed in the above sections, the communication performance involved in large batch training also plays an important role with respect to time-to-solution. The data parallel approach, where the global batch size is divided among model replicas, is one of the leading approaches to scale up. (The other approaches include model parallel and parallel hyper-parameter search.)

In Figure 8, we compare the 2 most popular distribution libraries for data parallel using PyTorch: 1) Horovod and 2) Distributed Data Parallel (DDP). Horovod, a third-party plugin, supports a novel communication pattern in which
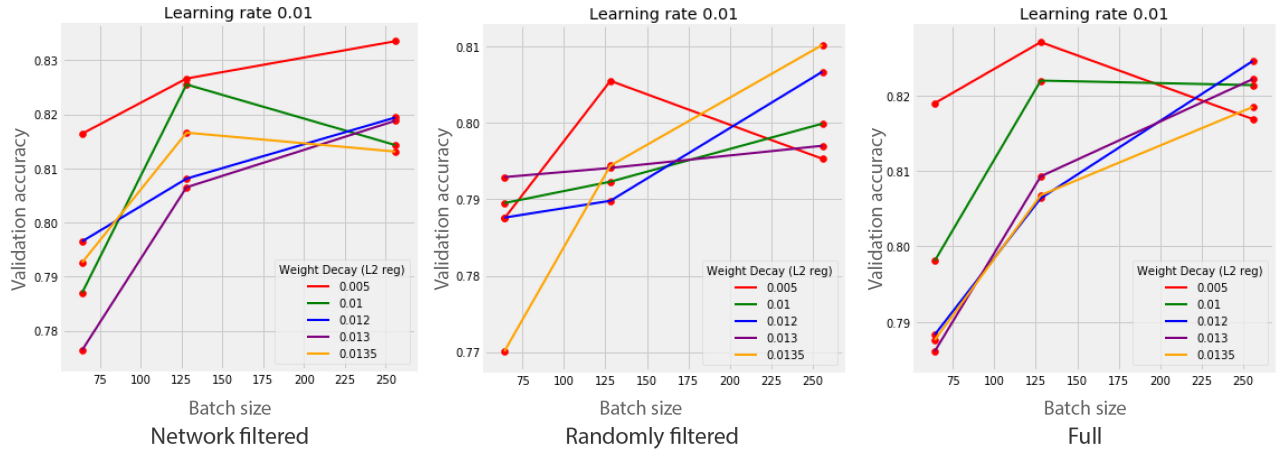
Fig. 6. Small batch size vs. Accuracy for network filtered, randomly filtered and full CIFAR-10 data sets.

tensors ready to be "allreduced" are placed into a queue (controlled by the buffer size and cycle time parameters) first, and depending on which set of tensors are ready the communications are scheduled dynamically. As shown in Figure 8(a) and (b), for one batch update, there is a long negotiation phase to fill up the tensor queue and the communication is divided into several allreduce calls to maximize the overlap between communication and computation (see Figure 8(a) 2 kernel streams). However, the overhead of dynamic queuing undermines its benefit, and the scaling deteriorates near 1000 GPUs. DDP, on the other hand, is a built-in library within PyTorch which handles the tensor reduction in static order and can also hide some communication cost by overlapping communication and computation (see Figure 8 (c)). For the scaling comparison in Figure 8, we use default parameters for buffer size and cycle time for Horovod, default message counts for DDP. Tuning Horovod parameters for 96 GPUs presents only marginal improvements and DDP demonstrates an overall superior scaling for large batch distributed training of ResNet models.

V. Conclusion

Large-scale deep neural networks require a large amount of computation to converge to acceptable testing accuracy. Research in efficient and scalable training mostly focuses on efficiently distributing the data to perform data-parallel computation on multiple nodes. However, data communication between machines in the large-scale system becomes the bottleneck of the system throughput. While using a large mini-batch size can improve the scalability and throughput of the system, it is hard to keep the generalization gap small when data is divided into batches. In this study, we empirically approach this problem from the input data perspective and propose a data optimized strategy for large batch distributed training. We use traditional machine learning techniques (t-SNE and DBSCAN) to refine data quality and remove data points that may be deemed to appear as noise in feature space. We find that for certain large batch sizes of about 6k and above we achieve better performance, and for batch sizes smaller
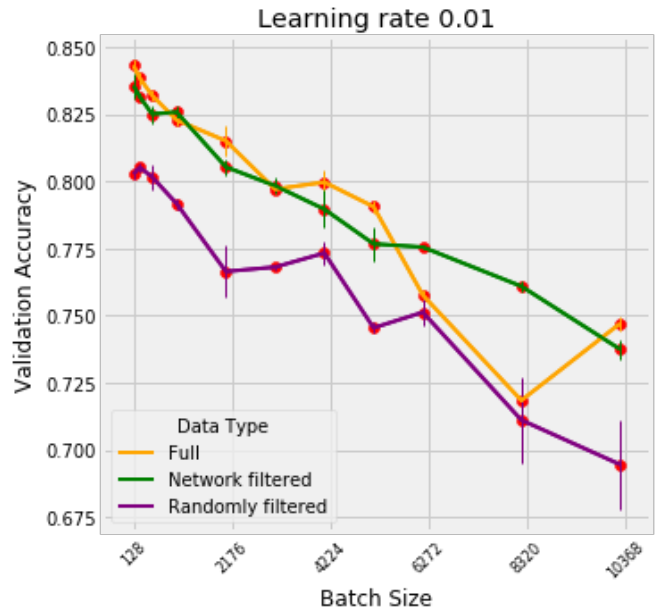

Fig. 7. Large batch training accuracy for Full CIFAR-10, Randomly filtered and Network filtered data (Avg. of 4 runs). Network filtered data gives at least the same accuracy as full data and outperforms randomly filtered data.

than 6k, we achieve similar accuracy compared to when a model is trained on all the data. For large-batch training in practice, we find that that the PyTorch built-in distributed data parallel messaging library scales better than Horovod for ResNet models trained on up to 1536 V100 GPUs.

With regards to future extensions of this work, it is worth considering performing hyper-parameter tuning for the DBSCAN and t-SNE algorithms instead of using the default values and using the technique recursively as described in Figure 1. This will give us a more robust clustering and filtering on the data set. Different sampling techniques like stratified grid sampling of the sample data points may prove sufficient instead of applying DBSCAN to the entire t-SNE output to remove noisy data. Our work lends itself to con-
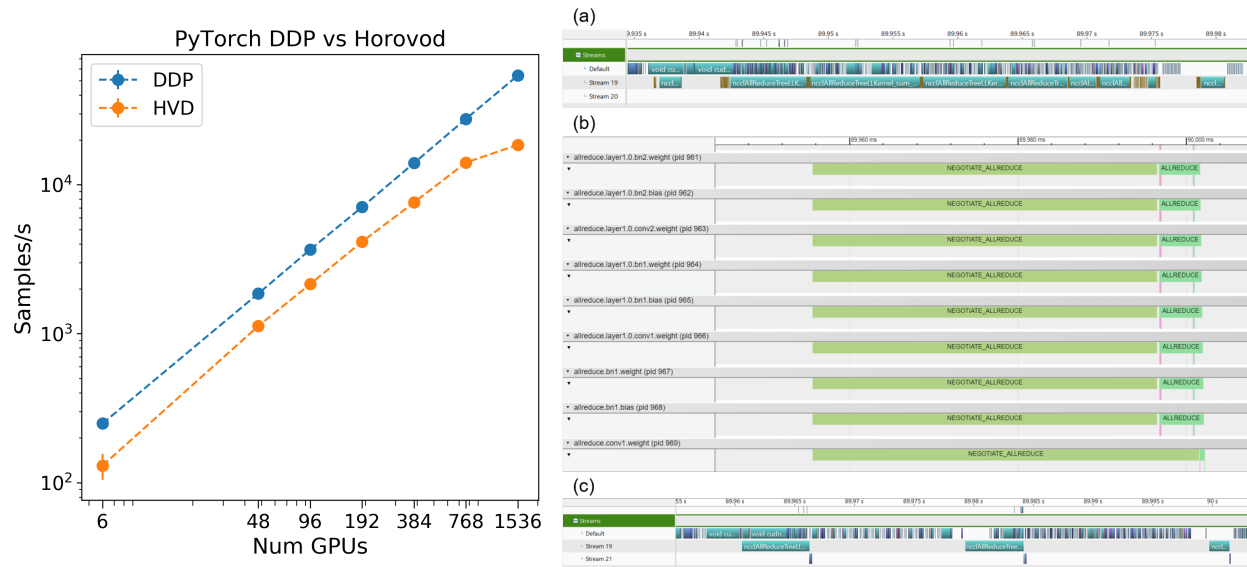
Fig. 8. (left) Data parallel scaling (effective batch size is linearly proportional to the number of GPUs) of PyTorch with built-in distributed data parallel (DDP) library vs third-party Horovod library, both using NCCL communication backend. (right) (a) nvprof profiling of a typical step in PyTorch with Horovod; (b) Horovod timeline tracing of a step; (c) nvprof profiling of a typical step in PyTorch with DDP.

version into an interactive tool to visualize and investigate the working of a model using such a methodology which can facilitate reproducibility and model transparency. We find that our technique works with a pre-trained ResNet-101 used as a feature extractor as well, instead of training a model from scratch. We intend to expand this work using other pre-trained networks in the future. These techniques are likely to be particularly effective in a variety of domain areas such as anomaly detection, data compression, and data filtering.

REFERENCES

[1] J. Yin et al. "Strategies to Deploy and Scale Deep Learning on the Summit Supercomputer". In: *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*. 2019, pp. 84–94.

[2] S. McCandlish et al. *An Empirical Model of Large-Batch Training*. 2018. eprint: arXiv:1812.06162.

[3] S. Smith et al. *Don't Decay the Learning Rate, Increase the Batch Size*. 2017. eprint: arXiv:1711.00489.

[4] E. Hoffer et al. "Train longer, generalize better: closing the generalization gap in large batch training of neural networks". In: (2017). eprint: arXiv:1705.08741.

[5] H. Li et al. *Visualizing the Loss Landscape of Neural Nets*. URL: https://papers.nips.cc/paper/7875-visualizing-the-loss-landscape-of-neural-nets.pdf.

[6] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Apr. 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[7] A. Sergeev et al. "Horovod: fast and easy distributed deep learning in TensorFlow". In: *arXiv preprint arXiv:1802.05799* (2018).

[8] M. Jogin et al. "Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning". In: *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. 2018, pp. 2319–2323.

[9] L. Maaten et al. *Visualizing Data using t-SNE*. URL: https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf.

[10] M. Ester et al. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. 1996. URL: https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf.

[11] T. He et al. *From Data Quality to Model Quality: an Exploratory Study on Deep Learning*. 2019. eprint: arXiv:1906.11882.

[12] X. Jia et al. *Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes*. 2018. eprint: arXiv:1807.11205.

[13] Y. You et al. *Large Batch Training of Convolutional Networks*. 2017. eprint: arXiv:1708.03888.

[14] S. Gahlot et al. *Data optimal large batch distributed training of Deep Neural Networks*. Version 1.0. Oct. 2020. URL: https://github.com/ghltshubh/dolbdtDNN.

[15] D. Masters et al. *Revisiting Small Batch Training for Deep Neural Networks*. 2018. eprint: arXiv:1804.07612.