Parallel and Distributed Training of Deep Neural Networks: A brief overview

Attila Farkas*, Gábor Kertész† and Róbert Lovas‡
*†‡Institute for Computer Science and Control (SZTAKI), Hungary
*Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, Hungary
*attila.farkas@sztaki.hu, †gabor.kertesz@sztaki.hu, ‡robert.lovas@sztaki.hu

***Abstract*—Deep neural networks and deep learning are becoming important and popular techniques in modern services and applications. The training of these networks is computationally intensive, because of the extreme number of trainable parameters and the large amount of training samples. In this brief overview, current solutions aiming to speed up this training process via parallel and distributed computation are introduced. The necessary components and strategies are described from the low-level communication protocols to the high-level frameworks for the distributed deep learning. The current implementations of the deep learning frameworks with distributed computational capabilities are compared and key parameters are identified to help design effective solutions.**

I. Introduction

Nowadays, machine learning, and in particular deep learning is a rapidly growing technology and taking over variety of aspect of our daily lives. The core component of the deep learning is the deep neural network, a structure inspired by the human brain. Deep neural networks could discover the connections in huge datasets to provide solutions for unsolved problems. Applications are available in natural language processing [1], image and video processing [2] [3] and text processing [4] as well. These networks also could be used in more extraordinary domains like weather [5] or volcano eruption forecasting [6]. For this functionality, the deep neural networks have to be trained on large amount of data to offer high performance solutions. Such training processes could be time consuming, therefore, different parallel and distributed solutions are needed in order to speed up the training process. In this paper, current solutions, architectures, popular frameworks and technological limitations of parallel and distributed deep learning are discussed.

A. Deep learning

A neural network [7] is a computation model which consists of many computation units, addressed as neurons. These neurons are organized into interconnected layers, which form the neural network. In the network the input neurons are activated from input parameters, the neurons of the following layer are activated through a weighted connection from neurons of the previous layer.

The neural network have to determine the correct weight value for every connection to provide the desired functionality, in most cases classification or regression. The network optimizes these weights during the training process. These weights are also referred to as trainable parameters.

In the classical terminology, there are two main types for machine learning, supervised and the unsupervised learning [8]. In the supervised learning, the model is trained via labeled input information. With the labels, the expected output is known during the training; therefore, at each training sample the error can be calculated, and the weights can be modified accordingly. On the other hand, unsupervised learning do not need labeled training data; the model works on its own to discover the related information and develop representational patterns. While researchers agree, that unsupervised learning will lead to breakthroughs in the future [9], the current methods give unpredictable, and in some cases unstable, results. In this paper we focus on supervised learning, which is a more mature technique in case of deep neural networks.

Neural networks were limited to a so-called shallow architecture with only a one, or a few hidden layers for many decades. These structures were able to solve basic classification and regression problems, but they were not capable to provide higher-level, complex functionalities. Therefore, the hidden layers and the neuron counts have to be increased to create more sophisticated and functional neural networks. These neural networks with more than one hidden layer are called deep neural networks. Because of the increased number of neurons, novel activation functions and optimization methods, new network architecture designs, deep neural networks are now capable to enable more complex decision and functionalities. However, the increased number of parameters, and the increased amount of training data have a significant negative effect on training time. In the following sections different approaches are introduced on aiming to speed up the training process of the deep neural networks.

B. Limitations of parallel execution

CPUs are capable to execute training process of a shallow neural network with acceptable training time because of the low number of trainable parameters. In the deep neural networks, the frequently occurring extreme number of trainable parameters and training samples define a need for large-scale parallel execution than the CPU could provide to keep the acceptable training time.

Therefore the GPU-based execution came up because of the massively parallel execution capabilities. The main advantage

978-1-7281-1059-2/20/$31.00 ©2020 IEEE

of the graphical accelerator-based architecture is the massive number of execution units. These units offer less functionality than a single CPU core, however in case of training neural networks using backpropagation basic operations like multiplication are adequate. Because of the large number of processing units in GPUs, and since the steps during the training process of a neural network are basic matrix and vector operations, the process can be efficiently parallelized using graphical processors. Thanks to this novel method of implementation [10], [11], [12], [13] deep learning started to gain popularity in the 2010s. In these years, other new procedures including novel activation functions, regularization methods and optimization algorithms were introduced to increase performance of the deep neural networks.

Parallel execution via GPUs has its limitations as well. Since deep neural networks need more training data compared to shallow networks to achieve high performance. Therefore, the size of the memory on GPU cards and the data transfer between the CPU and GPU memory also could be a bottleneck during the training process.

The creation of smaller batches from the whole training set is possible solution to this transfer capacity problem. The effective size of batches is also a question, because it could cause early overfitting, or underfitting on the training data. The finite computing capacity of one GPU card also could be a bottleneck during the training process.

The vertical scaling of the GPUs could resolve capacity issues. Nowadays, there are available implementation to use more GPU cards or more nodes with GPU cards to train the deep neural networks and the following sections will introduce the main implementations.

II. Distributed Deep Learning

Distributed deep learning enables the developers to use more than one GPU card during the training of a deep neural network. There are different strategies for implementation of the distributed architecture, where the abstraction of the nodes, on GPU level or on server level or the communication between the different nodes are different.

A. Communication protocols

To create a distributed deep learning framework, the first task is to select an efficient way to communicate between the GPU cards and the nodes [14].

NVLink [15] is communication interface developed by Nvidia for inter-GPU communication. The main advantage of NVLink is removing the limitation of PCIe connections and providing faster communication between the GPUs. NVLink enables bidirectional communication with 8 different pairs in each direction for a total of 32 wires. This connection is performed 20 Gbps data transfer in each direction and 40 Gbps bidirectional transfer speed. Nvidia V100 [16] utilise 6 NVLink connection to provide 300 Gbps inter-GPU communication meanwhile the standard PCIe Gen3 communication only provides 32Gbps transfer speed. NVLink could create a cluster from the connected GPU cards and offers this cluster like one executable unit for the developers. Therefore, the programmers do not have to manage the execution of tasks and memory allocation per GPUs in the system.

The Message Passing Interface (MPI) [17] is a language-independent communication standard for inter-process communication. MPI provides a language-independent communication between a set of processes which could be located on different systems as well. MPI functionality includes synchronous or asynchronous point-to-point communication via send and receive operations for communication and data transfer as well. MPI also offers broadcast functionality to populate information in the whole network. Furthermore, MPI facilitates gather and reduce operations to collect and aggregate information from different processes. For such inter-process communications MPI could provide information about the current network and the currently connected processes, as well.

The NVIDIA Collective Communications Library (NCCL) [18], is a CUDA [19] library for topology-aware multi-GPU communication. NCCL is designed for inter-process communication across different nodes like MPI but GPU based processes. Therefore the NCCL API is similar to the MPI but there are many differences. NCCL provides the following collective operations: AllReduce. Broadcast, Reduce, AllGather, ReduceScatter. For inter-GPU and inter-node communication, NCCL supports the following communication protocols: PCIe, NVLINK, InfiniBand Verbs, and IP sockets. Because of these functionalities, NCCL gain popularity in the distributed deep learning frameworks.

B. Strategies

For distributed deep learning there are two main different approaches in terms of parallel execution: the data-parallel and the model-parallel [20] method. The data-parallel implementation means, that during the training, each node has the same model, and a subset of the training data is used for each step. After executing the steps simultaneously, the nodes aggregate the calculated gradients for the next iteration. The model-parallel strategy divides the work of the neurons in each layer and allocates this work to different execution units. This process incurs additional communication after each layer between the devices and during the training, the current part of the input data have to stored in the memory on every device.

One of the first implementations of the distributed deep learning was the Theano-MPI [21]. Theano is an open-source Python library for developing and optimizing complex algorithms with mathematical expressions which involves multi-dimensional arrays operations as well. Theano depends on CUDA to utilise GPGPU resources for the mathematical operations. This functionality made it popular in the deep learning community. Theano-MPI leverage CUDA-aware MPI solution for communication between different GPU cards. Based on the CUDA-aware MPI, Theano-MPI could offer a scalable training framework with multi-node and multi-GPU support whit efficient inter-GPU parameter transfer.

Chainer [22] is a Python-based open-source neural network platform. Chainer supports neural network creation and modification to offer more flexible neural network architectures. Chainer uses NumPy for array manipulation and depends on CUDA to support GPU based execution. Chainer contains ChainerMN (Chainer Multi-Node) [23] to facilitate distributed deep learning. The current implementation uses data-parallel computation for the distributed training with synchronous model update between two following epoch. ChainerMN uses MPI for inter-node communication and the NVIDIA NCCL for inter-GPU communication.

Apache MXNet [24] is an open-source deep learning framework which allows mixing the symbolic and imperative programming. MXNet contains dynamic dependency scheduler which automatically parallelizes the symbolic and imperative operations in runtime. MXNet has deep integration with the following programming languages: Python, Scala, Julia, Clojure, Java, C++, R and Perl. Because of the language support and the symbolic execution, MXNet supports faster development and training in Python, then deploy the developed model in a different language for production. For distributed deep learning, MXNet support two different approaches, the data-parallel and the model parallel exaction. [25] In the data-parallel execution, the model is mirrored between the different host and every host work with the same model but on a different subset of the data during the training like Chainer. MXNet uses a key-value store for data exchange during the distributed training. After one batch processing, the workers push the gradient to the database and pull the updated gradient for the next batch processing. On the other hand, the model parallel training is support to train the different parts of the model in separated execution units. MXNet assigns the parameters of the different layers to different devices to provide the model parallel execution. With model parallel execution, MXNet focusing on LSTM recurrent networks. MXNet separates the following LSTM modules into different GPUs to provide the whole training process. Therefore none of the GPUs knows the whole network and does not have to update the model after one step like in the data-parallel training. The key component in that mode to specify the separation of the network to minimize the communication and balance the workload between the GPUs.

The Microsoft Cognitive Toolkit (CNTK) [26] is an open-source deep learning platform for commercial-grade distributed deep learning. CNTK describes the network with a series of computation via a directed graph. CNTK provides libraries for Python, C++ and C# but could be used as a standalone tool via BrainScript which is CNTK's model description language. CNTK also uses CUDA to utilize GPU recourse for the training process. CNTK also offers distributed deep learning functionalities. [27]. For inter-host communication, CNTK uses MPI like Theano-MPI and Chainer. CNTK provides data-parallel distributed learning with different SGD implementations.

PyTorch [28] is an open-source machine learning framework for research prototyping to production deployment. PyTorch based on Python and provides API for C++ and Java as well. PyTorch defines Tensors which is similar to the ndarrays in NumPy but Tensors also could be used on GPU, too. Although, PyTorch offers a conversion method between Torch Tensor and NumPy arrays for easier portability. For GPU execution, PyTorch also leverages on CUDA, therefore, the Tensor operations could be executed on GPU cards, too. PyTorch enables distributed deep learning with data-parallel execution with a mirrored model between nodes and distributed Stochastic Gradient Descent (SGD) implementation like Chainer. For distributed learning, PyTorch provides three different communication backends like Gloo, MPI and NCCL [29]. Gloo is recommended for distributed CPU learning and NCCL for distributed GPU learning.

TensorFlow [30], [31] is an open-source Python-based machine learning framework from Google. TensorFlow also applies a computational graph to execute operations. TensorFlow offers API for JavaScript, C++ and Java as well. It also uses CUDA to utilise GPU resources but supports Tensor Processing Unit (TPU) for hardware acceleration, too. TPU is directly designed for machine learning procedures by Google. For basic operations, TensorFlow uses tensor object and represent it as an n-dimensional array internally. TensorFlow provides complex, feature-rich functionalities for computation process with a low-level API. Therefore, there are some higher-level packages to provide specific functionality on top of TensorFlow. One of these packages is Keras, for neural networks. Keras is an open-source Python-based high-level API for neural networks. Keras is capable to run on top the TensorFlow, CNTK and Theano. From TensorFlow 2.0, Keras is integrated and maintained in the TensorFlow package. Therefore the last multi-backend release is Keras 2.3.0 which makes significant API changes to support TensorFlow 2.0, as well. Keras seamlessly support CPU and GPU based execution thanks for the TensorFlow backend. TensorFlow support distributed deep learning via different strategies and can be used with higher-level API like Keras, as well. TensorFlow enables data-parallel distributed learning like the previous solutions in this section. For inter-GPU communication, TensorFlow uses NCCL by default, but support any other third party implementation of all-reduce procedure and for inter-host communication, it offers gRPC and NCCL, too. TensorFlow follows two different approaches for parameter exchange between the mirrored models [32]. One implementation is the MirroredStrategy when the all-reduce aggregates the gradient across all devices. On the other hand, TensorFlow provides CentralStorageStrategy for synchronous parameter exchange. In this strategy [33], the variables of the model are located on the CPU and the computation process are distributed between the GPUs. These strategies are implemented for one host with a single or multi-GPU support and for multiple hosts with GPU resources, too.

Horovod [34] is an open-source distributed deep learning framework from Uber for TensorFlow, Keras, PyTorch, and Apache MXNet. The main goal of Horovod is to provide an easy to use framework for distributed learning and inte-

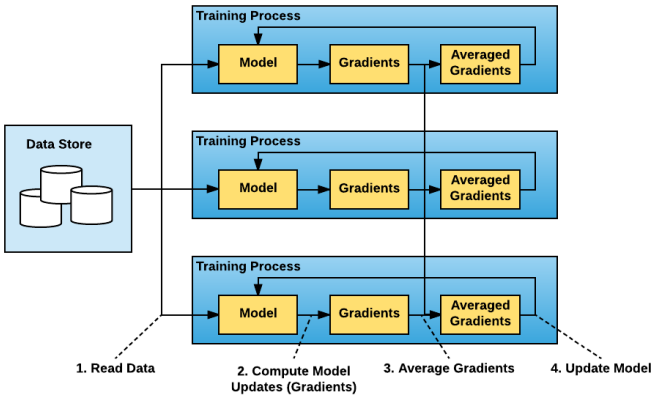

Fig. 1. All-reduce algorithm (source: [34])

grated with the main popular machine learning framework. Horovod support data-parallel distributed training whit multi-node support, but also support model-parallel approach but just within one node with multi-GPU support. Horovod follows a new approach for data-parallel learning. The original idea for gathering the gradients was the all-reduce procedure, which is presented in Figure 1. This is a synchronous operation and after one step every execution unit collects every gradient from the others, average them and then use this gradient for the next step. This algorithm generates N(N-1) communication after every step. The main problem with that implementation is the intensive network communication. After each step, every node has to change information and with higher node count the latency of that communication slow down the training process. Another implementation is to use a parameter server to gather the gradient and publish the average of them. This method could decree the necessary network communication but do not scale well, because with higher node count the parameter server could be a bottleneck in the system. The previous tools implement one of these methodologies for distributed learning.

The main purpose of Horovod is to provide a new communication format to gather the gradient to speed-up the learning process. Therefore, Horovod uses ring-allreduce strategy [35] for gradient exchange which is presented in Figure 2. This strategy only generates 2(N-1) communication between the nodes. In the first (N-1) iteration, every node sends to their neighbour a chunk of their data buffer and the receivers sum these chunks with their current data buffer value on the same index. After this iteration, every node will have one chunk, which stores the final sum of one data buffers. In the second (N-1) iteration every node send to their neighbour these final chunk values. At the end of these iterations, every node will sum of the data buffers. The optimal chunk size for this strategy depends on the current network connection between the hosts. The main objective for the chunk size is to reduce the network latency as possible. Horovod implements this all-reduce strategy via NCCL for inter-GPU communication across the nodes.

Because of these heterogeneous and diverse implementations of neural networks, the exchange and portability of the trained models are difficult tasks. This was the motivation and main driving force for the creation of the Open Neural Network Exchange (ONNX) [36]. ONNX is an open format to represent machine learning models. ONNX define a common set of operators and file format to provide the portability of the trained model between different frameworks. Therefore, the developers could use desired and training optimized framework for developing and training neural network models than could migrate their model into a production-grade framework for long-term execution.

The main properties of the introduced strategies are summarized in Table I. The data-parallel execution is supported by every strategy. For inter-GPU communication most of the implementations support Nvidia NCCL protocol, therefore, the NVLink connection supported as well. Every implementation offers ONNX functionality to model portability except Theano-MPI. Finally, Horovod supports a key-value feature for efficient distributed training because of the ring-allreduce strategy.

C. Performance evaluation

Several papers compare the above introduced inter-process communication solutions [37] and frameworks [38], [39] for distributed deep learning applications. These papers apply different types of deep neural network architectures and common datasets like CIFAR10 [40] and MNIST [41]. The difference between the various inter-process communication implementations is measured with the ResNet-50 [42] convolutional neural network.

For some of the measurements, the authors utilise up to 16 GPU cards to test the scalability of the communication protocols. The measurements deals with the Tensorflow and Horovod communication protocols, and the results [37] show that the ring-allreduce strategy of Horovod has the best performance indicators. Furthermore, the Horovod-NCCL2 communication achieved 10% speed-up comparing to the Horovod-MPI communication.

Concerning the frameworks, most of the papers compare the most popular frameworks such as Tensorflow, PyTorch and MXNet. The studies show that the introduced GPU-based deep learning offers better performance, but the modern CPU architectures could provide similar performance like the previous GPU generations [38]. Tensorflow and PyTorch have nearly the same scaling capability, but PyTorch utilizes better the GPU resources [38], on the other hand Tensorflow achieved better performance on distributed CPU resources [39].

D. Limitations

There are several different parameters which could influence the training time and the efficiency of the neural network, especially in distributed training. Larger batch size generally improves the scalability, because more work could be done between communication steps, therefore, the negative effect of the network latency could be decreased [43] [44]. However, the memory size of one execution unit is the limit of one batch size, and the increased batch size also could cause overfitting

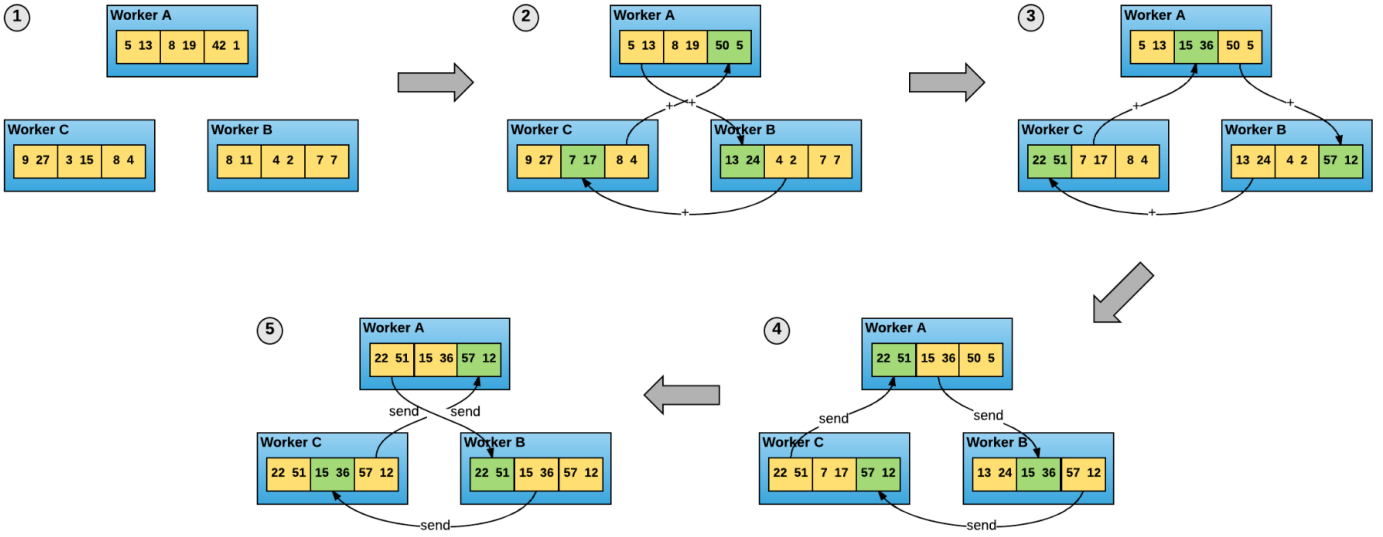

Fig. 2. Ring-allreduce algorithm (source: [34])

TABLE I
COMPARISON OF DISTRIBUTED DEEP LEARNING STRATEGIES

Functions\Tools	**TensorFlow** [30]	**PyTorch** [28]	**CNTK** [26]	**Apache MXNet** [24]	**Chainer** [22]	**Theano-MPI** [21]
Parallel execution	data-parallel	data-parallel, model-parallel	data-parallel	data-parallel, model-parallel	data-parallel, model-parallel	data-parallel
Communication protocol	Nvidia NCCL, gRPC	Nvidia NCCL, MPI, Gloo	Nvidia NCCL, MPI	Central key-value store	Nvidia NCCL, MPI	CUDA-aware MPI
Horovod support	X	X	-	X	-	-
ONNX support	X	X	X	X	X	-

in the network. Increasing the neural network size generally increases the accuracy of the network, but this could be cause communication overhead so the scalability decreases, and the memory size of the execution unit is the hard limit for the network size. The last influence factor for these parameters is the actual application. Because the application also could define constraints for the network topology or the effective batch size as well. These described constraints and limitations have to be resolved in each distributed deep learning strategy (see in Section II-B).

III. CONCLUSION AND FUTURE WORK

In this paper, we briefly introduced the main purpose of the distributed deep learning strategies and current limitations of deep learning in general. We described the necessary communication protocols for inter-GPU and inter-node communication. On top of these protocols, the current implementations of distributed deep learning frameworks have been compared. Finally, we described Horovod, which is one of the most popular distributed deep learning framework. Leveraging on the findings, we discussed the main limitations of the current distributed training frameworks.

Based on these results a methodology for efficient distributed deep learning, architecture selection and hyperparameter tuning are to be designed as the part of our future research plan.

IV. ACKNOWLEDGMENT

This work was partially funded by the European "Novel EOSC services for Emerging Atmosphere, Underwater and Space Challenges" (NEANIAS) project, Grant Agreement No. 863448 (H2020-INFRAEOSC-2019-1), and the National Research, Development and Innovation Office (NKFIH) under OTKA Grant Agreement No. K 132838. The authors thankfully acknowledge the support of the Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University.

REFERENCES

[1] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[5] M. A. Zaytar and C. El Amrani, "Sequence to sequence weather forecasting with long short-term memory recurrent neural networks," *International Journal of Computer Applications*, vol. 143, no. 11, pp. 7–11, 2016.

[6] E. A. Justin Parra, Olac Fuentes and V. Kreinovich, "Use of machine learning to analyze and – hopefully – predict volcano activity," *Acta Polytechnica Hungarica*, vol. 14, no. 3, pp. 209–221, 2017.

[7] J. Schmidhuber, "Deep learning in neural networks: An overview," *CoRR*, vol. abs/1404.7828, 2014. [Online]. Available: http://arxiv.org/abs/1404.7828

[8] A. I. Károly, R. Fullér, and P. Galambos, "Unsupervised clustering for deep learning: A tutorial survey," *Acta Polytechnica Hungarica*, vol. 15, no. 8, pp. 29–53, 2018.

[9] S. Tan, G. Androz, A. Chamseddine, P. Fecteau, A. Courville, Y. Bengio, and J. P. Cohen, "Icentia11k: An unsupervised representation learning dataset for arrhythmia subtype discovery," 2019.

[10] D. Steinkraus, I. Buck, and P. Simard, "Using gpus for machine learning algorithms," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE, 2005, pp. 1115–1120.

[11] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[12] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 873–880.

[13] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[14] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *arXiv preprint arXiv:1807.11205*, 2018.

[15] D. Foley and J. Danskin, "Ultra-Performance Pascal GPU and NVLink Interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, 2017.

[16] NVIDIA, "NVIDIA V100 Tensor Core GPU," https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf, 2020.

[17] W. Gropp, W. D. Gropp, E. Lusk, A. D. F. E. E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[18] NVIDIA, "NVIDIA Collective Communication Library (NCCL) Documentation," https://docs.nvidia.com/deeplearning/sdk/nccl-developer-guide/docs/index.html, 2020.

[19] ——, "CUDA Toolkit Documentation v10.2.89," https://docs.nvidia.com/cuda/, 28 Nov 2019.

[20] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[21] H. Ma, F. Mao, and G. W. Taylor, "Theano-MPI: a Theano-based Distributed Training Framework," *arXiv preprint arXiv:1605.08325*, 2016.

[22] S. Tokui, K. Oono, S. Hido, and J. Clayton, "Chainer: a next-generation open source framework for deep learning," in *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, vol. 5, 2015, pp. 1–6.

[23] T. Akiba, "Performance of distributed deep learning using chainermn," https://chainer.org/general/2017/02/08/Performance-of-Distributed-Deep-Learning-Using-ChainerMN.html, 8 Feb 2017.

[24] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[25] Apache, "Distributed Training in MXNet," https://mxnet.apache.org/api/faq/distributed_training, 2018.

[26] F. Seide, "Keynote: The computer science behind the microsoft cognitive toolkit: an open source large-scale deep learning toolkit for windows and linux," in *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2017, pp. xi–xi.

[27] Microsoft, "Multiple GPUs and Machines," https://docs.microsoft.com/en-us/cognitive-toolkit/multiple-gpus-and-machines, 2017.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[29] A. Jain, A. A. Awan, H. Subramoni, and D. K. Panda, "Scaling tensorflow, pytorch, and mxnet using mvapich2 for high-performance deep learning on frontera," in *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*. IEEE, 2019, pp. 76–83.

[30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[31] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[32] E. Bisong, "Tensorflow 2.0 and keras," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 347–399.

[33] S.-T. Wang, F.-A. Kuo, C.-Y. Chou, and Y.-B. Fang, "Performance evaluation of distributed deep learning frameworks in cloud environment," *International Journal of Computer and Information Engineering*, vol. 13, no. 8, pp. 439–443, 2019.

[34] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[35] A. Gibiansky, "Bringing hpc techniques to deep learning," http://andrew.gibiansky.com/, 2017.

[36] The Linux Foundation, "ONNX," https://onnx.ai/get-started.html, 2019.

[37] A. A. Awan, J. Bédorf, C. Chu, H. Subramoni, and D. K. Panda, "Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 498–507.

[38] A. Jain, A. A. Awan, Q. Anthony, H. Subramoni, and D. K. D. Panda, "Performance characterization of dnn training using tensorflow and pytorch on modern clusters," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 1–11.

[39] A. Jain, A. A. Awan, H. Subramoni, and D. K. Panda, "Scaling tensorflow, pytorch, and mxnet using mvapich2 for high-performance deep learning on frontera," in *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*, 2019, pp. 76–83.

[40] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[41] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[43] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[44] A. Koliousis, P. Watcharapichat, M. Weidlich, L. Mai, P. Costa, and P. Pietzuch, "CROSSBOW: scaling deep learning with small batch sizes on multi-gpu servers," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1399–1412, 2019.