# Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools

RUBEN MAYER and HANS-ARNO JACOBSEN, Technical University of Munich

Deep Learning (DL) has had an immense success in the recent past, leading to state-of-the-art results in various domains, such as image recognition and natural language processing. One of the reasons for this success is the increasing size of DL models and the proliferation of vast amounts of training data being available. To keep on improving the performance of DL, increasing the scalability of DL systems is necessary. In this survey, we perform a broad and thorough investigation on challenges, techniques and tools for scalable DL on distributed infrastructures. This incorporates infrastructures for DL, methods for parallel DL training, multi-tenant resource scheduling, and the management of training and model data. Further, we analyze and compare 11 current open-source DL frameworks and tools and investigate which of the techniques are commonly implemented in practice. Finally, we highlight future research trends in DL systems that deserve further research.

3

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Computer systems organization** → *Parallel architectures*; *Distributed architectures*;

Additional Key Words and Phrases: Deep-learning systems

**ACM Reference format:**
Ruben Mayer and Hans-Arno Jacobsen. 2020. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. *ACM Comput. Surv.* 53, 1, Article 3 (February 2020), 37 pages.
https://doi.org/10.1145/3363554

## 1 INTRODUCTION

Deep Learning (DL) has recently gained a lot of attention due to its superior performance in tasks like speech recognition [65, 69], optical character recognition [20], and object detection [95]. The application of DL poses a tremendous potential in numerous areas like medical image analysis (e.g., breast cancer metastases detection) [107], machine translation [84], image restoration (e.g., automatically colorize grayscale images) [75], image captioning [68] (i.e., creating a description of an image), and as agents in reinforcement learning systems that map state-action pairs to expected rewards [10]. In DL, a network of mathematical operators is trained with classified or unclassified data sets until the weights of the model are ready to make correct predictions on previously unseen data. Major companies and open-source initiatives have developed powerful DL frameworks such as TensorFlow [4] and MXNet [125] that automatically manage the execution of large DL models developed by domain experts.

Authors' address: R. Mayer and H.-A. Jacobsen, Technical University of Munich, Boltzmannstrasse 3, Garching, 85748, Germany; emails: {ruben.mayer, arno.jacobsen}@tum.de.

0360-0300/2020/02-ART3 $15.00
https://doi.org/10.1145/3363554

One of the driving factors of the success of DL is the scale of training in three dimensions. The first dimension of scale is the size and complexity of the models themselves. Starting from simple, shallow neural networks, with increasing depth and more sophisticated model architectures, new breakthroughs in model accuracy were achieved [30, 38]. The second dimension of scale is the amount of training data. The model accuracy can, to a large extent, be improved by feeding more training data into the model [56, 63]. In practice, it is reported that 10s to 100s of Terabyte (TB) of training data are used in the training of a DL model [27, 62]. The third dimension is the scale of the infrastructure. The availability of programmable highly parallel hardware, especially graphics processing units (GPUs), is a key-enabler to training large models with a lot of training data in a short time [30, 206].

Our survey is focused on challenges that arise when managing a large, distributed infrastructure for DL. Hosting a large amount of DL models that are trained with large amounts of training data is challenging. This includes questions of parallelization, resource scheduling and elasticity, data management and portability. This field is now in rapid development, with contributions from diverse research communities, such as distributed and networked systems, data management, and machine learning. At the same time, we see a number of open-source DL frameworks and orchestration systems emerging [4, 24, 141, 195]. In this survey, we bring together, classify, and compare the huge body of work on distributed infrastructures for DL from the different communities that contribute to this area. Furthermore, we provide an overview and comparison of the existing open-source DL frameworks and tools that put distributed DL into practice. Finally, we highlight and discuss open research challenges in this field.

## 1.1 Complementary Surveys

There are a number of surveys on DL that are complementary to ours. Deng [41] provides a general survey on DL architectures, algorithms, and applications. LeCunn et al. provide a general overview of DL [95]. Schmidhuber [156] provides a comprehensive survey on the history and technology of DL. Pouyanfar et al. [143] review current applications of DL. Luo [109] provides a review on hyper-parameter selection strategies in ML training, including training of neural networks. Those surveys cover general techniques of DL, but are not focused on scalability and distributed systems for DL.

Ben-Nun and Hoefler [14] provide an analysis of concurrency in parallel and distributed DL training. Chen and Lin [25] provide a survey on DL challenges and perspectives with regard to Big Data (i.e., high data volumes, variety and velocity). Erickson et al. [45] provide a short overview of DL frameworks. Our survey takes a much broader view on distributed DL systems. In particular, we include topics such as resource scheduling, multi-tenancy and data management. Those aspects of scalable DL systems become particularly important when dealing with large models and huge amounts of training data in a shared cluster or cloud environment. Furthermore, we analyze current open-source DL frameworks and tools in depth and relate them to the research on parallel and distributed DL training. This has not been done in the existing surveys. Pouyanfar et al. [143] analyze and compare DL frameworks, but not with regard to parallelization and distribution.

## 1.2 Structure of the Survey

We structure our survey as follows. In Section 2, we introduce DL and provide the foundations for the further discussion of DL systems. In Section 3, we discuss the challenges and techniques of scalable DL in detail. We cover four important aspects: Distributed infrastructures, parallelization of DL training, resource scheduling and data management. In Section 4, we analyze and compare 11 open-source DL frameworks and tools that put scalable DL into practice. Finally, in Section 5,

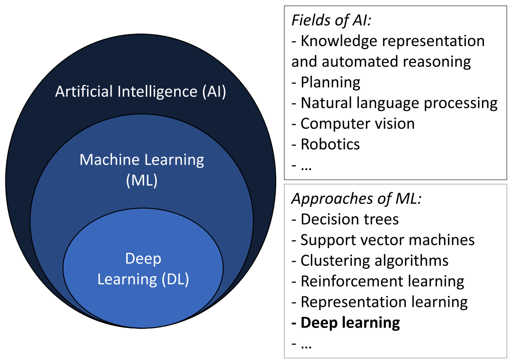

Fig. 1. Relationship between AI, ML, and DL.

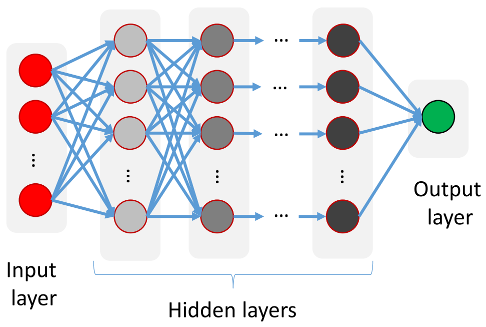

Fig. 2. Schematic of a multi-layer perceptron.

we conclude this survey and provide an outlook on current trends and open problems in the field that deserve further research.

## 2 FOUNDATIONS

### 2.1 Context of Deep Learning

Artificial intelligence (AI) has been a long held vision of building and programming computers in such a way that they can independently (i.e., without human involvement) solve complex problems [131, 157]. In the most recent past, immense practical achievements of AI have been made in many different fields, such as knowledge representation and automated reasoning [165], planning [87], natural language processing [198], computer vision [169], and robotics [99]. Among the methods developed in AI research are cybernetics, symbolic and sub-symbolic, and statistical machine learning (ML). Deep Learning (DL) is a specific approach of ML, which deals with the training of *deep neural networks*. The relationship between AI, ML, and DL is visualized in Figure 1.

### 2.2 Deep Neural Networks

A neural network (NN) is a network of interconnected *artificial neurons*, which are mathematical functions that transform a set of input signals to an output signal. By layering the neurons and connecting them from an input layer to an output layer, the overall network represents a function $f : x \to y$ that maps the input signals that go into the input layer (layer 1) to an output signal that leaves the output layer (layer $n$). The goal of $f$ is to approximate a *target function* $f^*$, e.g., a classifier $y = f^*(x)$ that maps an input $x$ to a category $y$. In the *training process*, the set of parameters $\Theta$, i.e., the weights, biases, and thresholds, in all of the artificial neurons are adjusted in such a way that the output of $f$ approximates the output of $f^*$ with the best possible accuracy. This is commonly achieved by applying *back-propagation* [152] to the gradient of the loss function w.r.t. the weights of the corresponding layers. There are different gradient descent algorithms applied in DL; a detailed review of gradient descent algorithms is provided by Ruder [151]. In the training process, instead of single training samples, *mini-batches* of training data are used in each iteration. This has the advantage of increased parallelism in the training process: The output of the network can be computed for a whole batch of training samples in parallel. However, choosing too large mini-batch sizes may deteriorate the model accuracy and increases the memory footprint of the training process [112]. The parameters of the training process itself, i.e., the loss function, gradient descent algorithm, activation function, step size (the factor by which the weights are changed toward the gradient), and size of the mini-batches are called *hyper-parameters*.

### 2.3 Neural Network Architectures

The simplest way of organizing a DNN is by using multiple fully connected layers of neurons, i.e., each neuron in a layer is connected to each neuron in the subsequent layer. This architecture is also referred to as *multi-layer perceptron* (MLP) (cf. Figure 2). However, MLPs have limitations [53, 96]. First, MLPs have a large number of weights, which requires a large number of training samples and occupies a large amount of memory. Second, MLPs are not robust against geometric translations and local distortions of the inputs. For instance, in the detection of hand-written digits from images, the same digit will be written slightly different in different images [96]. Third, MLPs are agnostic to the topology of the input, i.e., the order of the input signals is not taken into account. However, in many cases, there is a local structure in the input data. For instance, in images, pixels that are nearby are likely to be correlated [96], and in speech recognition, previous and future context of the input data is particularly relevant to detect a spoken word [53]. To overcome the shortcomings of MLPs, more sophisticated neural network architectures have been proposed. Here, we briefly review the most prominent ones.

Convolutional neural networks (CNNs) [96] introduce convolutional layers and sub-sampling layers. Different from fully connected layers as in MLPs, convolutional layers are only connected to sub-areas of their respective previous layers, pursuing the concept of *local receptive fields*, which is inspired by biology [72]. A convolutional layer is composed of multiple planes, where in each plane, all neurons share the same weights (*weight sharing*). Finally, convolutional layers alternate with *sub-sampling* layers to reduce the spacial resolution of the feature map. Besides feed-forward networks (where the output of neurons does not loop back to their own input), loop-backs are useful for many use-cases. For instance, in natural language processing, the meaning of one word in a sentence may depend on the meaning of a previously seen word in the same (or even a previous) sentence. To model such phenomena in DL networks, recurrent neural networks (RNNs) have been proposed. Long-short term memory (LSTM) units are special units of an RNN to overcome issues of exploding or vanishing gradients when training RNNs [67]. Autoencoders [66] are NNs that are used to learn efficient encodings (i.e., compressed representations) that extract significant features from the training data. Their architecture consists of an encoder, a code, and a decoder, each consisting of layers of neurons, where the output layer of the network has the same number of neurons as the input layer, but the code, which is exactly between encoding and decoding layers, has much fewer neurons. In generative adversarial networks (GANs) [51], two NNs are aligned with each other, namely, a generative and a discriminative NN. Another recent architecture of NNs are *graph neural networks* [192], where graph-structured representations are learned, as opposed to representations in the Euclidian space (as in CNNs).

## 3 DISTRIBUTED DEEP LEARNING

Training large DL models with vast amounts of training data is a non-trivial task. Often, it is performed in a distributed infrastructure of multiple compute nodes, each of which may be equipped with multiple GPUs. This brings a number of challenges. First, the processing resources must be effectively used, i.e., one must avoid stalling of costly GPU resources due to communication bottlenecks. Second, the compute, storage and network resources are typically shared among different users or training processes to reduce costs and provide elasticity (i.e., the cloud computing paradigm [9]). To tackle those challenges in DL, research at the intersection of computing systems and DL is receiving growing attention [4, 27, 36, 79, 141, 195]. This becomes evident with new workshops and conferences arising that particularly focus on DL/ML systems research, such as the *Conference on Systems and Machine Learning* (SysML).[1] However, also established communities

[1]https://www.sysml.cc.

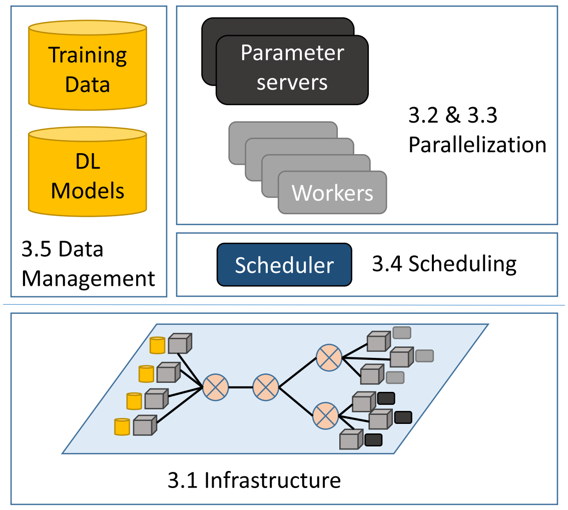

Fig. 3. Overview.

such as the data management community are turning their attention toward DL/ML systems [93, 185]. In this section, we discuss the main directions of DL systems research in depth. We introduce the main research challenges, discuss state-of-the-art approaches, and analyze open research problems that deserve further attention.

**Section Overview.** Figure 3 provides an overview of the topics addressed in this section. On the lowest level, we address the infrastructure used in large DL systems in Section 3.1. We cover recent trends in the hardware being used, networking architectures, as well as low-level software architecture for DL systems. On a higher level, we discuss methods for parallel DL training in Section 3.2. In Section 3.3, we more specifically discuss challenges and approaches for data-parallel training. To map the components of a parallel DL system to the infrastructure, scheduling is applied. In Section 3.4, we discuss the scheduling problem in single-tenant as well as multi-tenant scenarios. One of the big challenges of large-scale DL is the size of training data and DL models that need to be maintained. In Section 3.5, we discuss challenges and approaches of data management in DL.

## 3.1 Infrastructure

To understand the challenges on parallelization, scheduling and data management for DL, we first take a deeper look at the infrastructure on which DL training is performed. We divide the existing work into two categories: Hardware innovations and data-center scale infrastructure applied to real DL workloads. While the former can potentially be used on single compute nodes or small clusters, the latter describes how individual hardware components can be composed into a scalable, distributed infrastructure for DL.

*3.1.1 Hardware Components for DL.* While early DL deployments were based on clusters of multi-core CPUs, scalability limitations pushed the efforts to exploiting highly parallel hardware, and even developing special-purpose hardware dedicated to DL training and serving. The performance benefits of GPUs compared to CPU depend on many factors, such as whether the job is processing-bound or memory-bound, the efficiency of the implementation, as well as the hardware itself [97]. Both CPUs and GPUs hardware innovates at a fast pace, which makes comparisons difficult and short-living. Nevertheless, state-of-the-art infrastructures for DL typically comprise

GPUs to accelerate the training and inference process. Hardware vendors offer specialized servers and even workstations for DL, such as NVIDIA DGX station [2].

Besides GPU-centric DL, other forms of hardware acceleration have been proposed, such as field-programmable gate arrays (FPGAs) [135]. One strength of FPGAs that is repeatedly mentioned is their capability to make DL training and inference more energy-efficient. NeuFlow by Farabet et al. [46] is one of the first works that tackled the problem of using FPGAs for DL, in particular, for vision systems. Caffeine by Zhang et al. [201] is a hardware and software co-designed library to support CNNs on FPGAs. On the hardware side, it provides a high-level synthesis implementation of an FPGA accelerator for CNNs. In their design, they build upon previously developed methods such as unrolling and pipelining (cf. Zhang et al. [202]). On the software side, Caffeine provides a driver that allows for easily integrating FPGAs. Caffeine has been integrated into the Caffe DL framework and shows a reduction of energy consumption of up to 43.5× compared to CPU and up to 1.5× compared to GPU execution. Wang et al. [181] propose a custom FPGA design, called DLAU, to support the training of deep neural networks. One major challenge they had to overcome is the limited memory capacity of FPGAs. They propose tile techniques to partition the training data, along with FIFO buffers and pipelined processing units to minimize memory transfer. In their evaluations, they show that DLAU can train neural networks with up to 10x less energy consumption than GPUs. Tensor processing units (TPUs) are application-specific integrated circuits (ASICs) developed by Google that speed-up DL training and inference significantly [86]. TPUs are proprietary and not commercially available, but can be rented via the Google cloud services.

Besides such more traditional forms of computing architectures that follow the von-Neumann architecture by separating memory and processing units, there are research efforts to develop novel in-memory computing architectures (also called *neuromorphic hardware* [21]). Those efforts are inspired by the physiology of the brain, which is very different from the way traditional von-Neumann computing architectures work. Neurostream by Azarkhish et al. [11] is a processor-in-memory solution that is tailored toward training CNNs. However, neuromorphic hardware architectures are still in the experimental stage and not widely available.

Some papers have highlighted the need for efficient implementations of DL kernels, e.g., by exploiting SIMD (single instruction, multiple data) instructions [97, 176] and awareness of non-uniform memory access (NUMA) [150]. This raises the need for re-usable, optimized kernel implementations of the most relevant operations in DNN training. One of the major GPU-specific libraries is cuDNN, a library with DL primitives for GPUs [26]. The NVIDIA Collective Communications Library (NCCL) [1] provides multi-GPU and multi-node communication primitives and is optimized for PCIe and NVLink high-speed interconnects. DL frameworks often incorporate such low-level libraries to fully exploit the capabilities of the hardware infrastructure.

*3.1.2 Large-scale Infrastructure for DL.* A large-scale DL infrastructure is composed of many inter-connected hardware components that together build a *warehouse-scale computer* [13]. In this subsection, we review current infrastructures as described by organizations that perform very large DL jobs, such as Facebook, Google, and Microsoft, as well as academic research.

Facebook describes its ML infrastructure in a recent paper [62]. They use both CPUs and GPUs for training, and rely on CPUs for inference. To do so, they build specialized CPU-based and GPU-based compute servers to serve their specific needs of training and inference. For training, GPUs are preferred, as they perform better; however, in their data centers, they have abundant capacities of readily available CPUs, especially during off-peak hours, which they also exploit. For inference, they rely on CPUs, as GPU architectures are optimized for throughput over latency, but latency is a critical factor in inference. Interestingly, for inter-connecting training servers in distributed, data-parallel training, they rely on 50G Ethernet, and forego using specialized interconnects such as RDMA or NCCL [1].

Similarly to Facebook, Tencent employs a heterogeneous infrastructure with both CPUs and GPUs. Their deep-learning system Mariana [211] consists of three different frameworks that are optimized for different infrastructures and use cases.

Adam is a large-scale distributed system for DL at Microsoft [27]. It relies on a large number of commodity hardware CPU-servers to perform DL training. Besides many system-level optimizations, one of the hardware-centric features of Adam is that they partition DL models in such a way that the model layers fit in the L3 cache to improve training performance.

The paper on TensorFlow [4], a scalable ML framework developed by Google, provides some insights into the infrastructure at Google. Overall, Google follows a different approach from Facebook and Microsoft when it comes to the DL infrastructure. First, they employ TPUs, which are custom ASICs, as opposed to only using commercial-off-the-shelf (COTS) hardware. Second, they exploit specialized interconnects and use multiple communication protocols, such as gRPC over TCP and RDMA over Converged Ethernet (RoCE).[2] Distributed TensorFlow supports communication via the message passing interface (MPI) [180].

In academic research, exploiting high-performance computing (HPC) infrastructures for DL training is a topic of increasing importance. Coates et al. [32] report using a cluster of 16 servers, each equipped with two quad-core CPUs and 4 GPUs, being interconnected by Infiniband. Different from Ethernet, Infiniband has high throughput and—more important—extremly low end-to-end latency (in the order of microseconds). Ben-Nun and Hoefler [14] also observe a trend to move towards HPC infrastructures in DL research.

Summing up, large-scale infrastructures in real-world deployments are highly heterogeneous. They do not only comprise GPU servers, but commonly also CPUs. Overall, we see a certain dominance of COTS hardware, just as it is also the case in other Big Data analytics workloads, such as batch processing [39] and graph processing [111]. However, also custom hardware and HPC infrastructure is used, especially at Google and in academic research. In HPC infrastructures, we observe that the DL systems are specialized toward the target infrastructures to increase performance, e.g., regarding the communication protocols like RDMA, NCCL, and MPI.

Performance of distributed infrastructures can be measured, e.g., in terms of throughput, latency and energy consumption. Besides the raw maximum performance of the hardware, another important factor is the communication protocol, e.g., whether RDMA is used. Further important questions are how the hardware components are composed to avoid bottlenecks. Li et al. [100] have performed a comprehensive performance evaluation of recent GPU interconnects. In terms of energy consumption, Wang et al. [181] provide evaluations that compare FPGAs to GPUs.

## 3.2 Parallelization Methods

DL comes with many possibilities for parallelization. Here, we introduce the three predominant parallelization methods in DL, namely data, model and pipeline parallelism, as well as hybrid forms of parallelism.

*3.2.1 Data Parallelism.* In data parallelism, a number of workers (machines or devices, e.g., GPUs) loads an identical copy of the DL model (see Figure 4). The training data is split into non-overlapping chunks and fed into the model replicas of the workers for training. Each worker performs the training on its chunk of training data, which leads to updates of the model parameters. Hence, the model parameters between the workers need to be synchronized. There are many

[2]RoCE is a network protocol that supports Ethernet as the underlying protocol for remote direct memory access (RDMA).

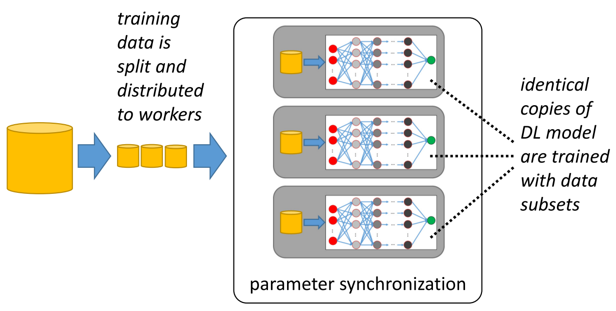
training data is split and distributed to workers
identical copies of DL model are trained with data subsets
parameter synchronization

Fig. 4. Data parallelism.

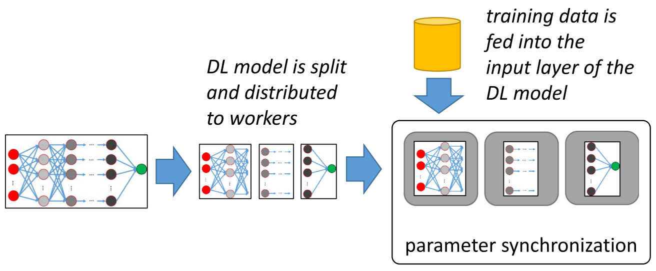
DL model is split and distributed to workers
training data is fed into the input layer of the DL model
parameter synchronization

Fig. 5. Model parallelism.

challenges in the problem of parameter synchronization. We discuss those challenges and state-of-the-art approaches to tackle them in Section 3.3.

The main advantage of data parallelism is that it is applicable to any DL model architecture without further domain knowledge of the model. It scales well for operations that are compute-intensive, but have only few parameters, such as CNNs. However, data parallelism is limited for operations that have many parameters, as the parameter synchronization becomes the bottleneck [82, 91]. This problem could be alleviated by using larger batch sizes; however, this increases data staleness on the workers and leads to poor model convergence. A further limitation of data parallelism is that it does not help when the model size is too large to fit on a single device. It is worth to note that in many data parallel training schemes, it is assumed or required that the training data is independent and identically distributed *(i.i.d.)*, so that parameter updates computed by the parallel workers can simply be summed up to compute the new global model parameters [196].

*3.2.2 Model Parallelism.* In model parallelism, the DL model is split, and each worker loads a different part of the DL model for training (see Figure 5). The worker(s) that hold the input layer of the DL model are fed with the training data. In the forward pass, they compute their output signal which is propagated to the workers that hold the next layer of the DL model. In the backpropagation pass, gradients are computed starting at the workers that hold the output layer of the DL model, propagating to the workers that hold the input layers of the DL model.

A major challenge of model parallelism is how to split the model into partitions that are assigned to the parallel workers [113]. A common approach to find a good model splitting is to use reinforcement learning [117, 118]: Starting from some initial partitioning, permutations on that partitioning are performed, and performance is measured (e.g., for one training iteration). In case of an improvement, the permutation is maintained, and further permutations are performed, until the measured performance converges. Streaming rollout [47] is a specialized solution that only works for RNNs.

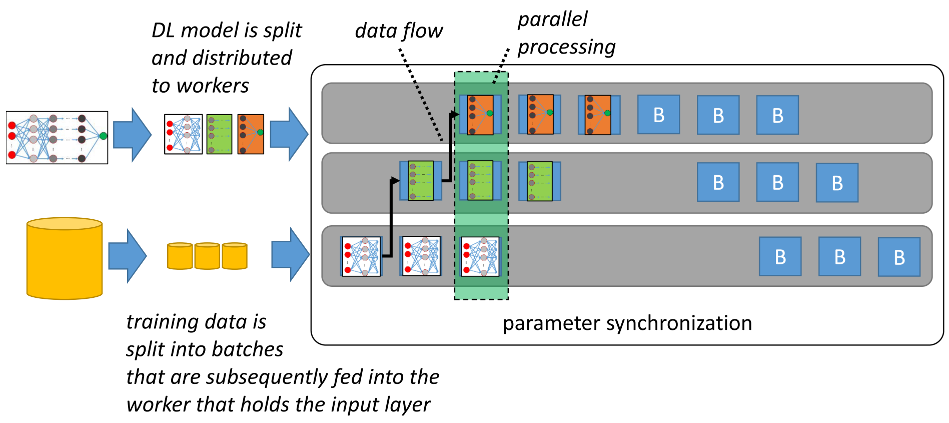
DL model is split and distributed to workers
data flow
parallel processing
B
training data is split into batches that are subsequently fed into the worker that holds the input layer
parameter synchronization

Fig. 6. Pipeline parallelism. "B"—Backpropagation. Figure adapted from Huang et al. [70].

The main advantage of model parallelism is the reduced memory footprint. As the model is split, less memory is needed for each worker. This is useful when the complete model is too large to fit on a single device. This can be the case when the device consists of specialized hardware such as GPUs or TPUs. The disadvantages of model parallelism are in the heavy communication that is needed between workers. As DL models are hard to be split effectively, there may occur stalling of workers due to communication overhead and synchronization delays. Hence, increasing the degree of model parallelism does not necessarily lead to training speedup [118].

*3.2.3 Pipeline Parallelism.* Pipeline parallelism combines model parallelism with data parallelism. In pipeline parallelism, the model is split and each worker loads a different part of the DL model for training (see Figure 6). Further, the training data is split into microbatches. Now, every worker computes output signals for a set of microbatches, immediately propagating them to the subsequent workers. In the same way, in the backpropagation pass, the workers compute gradients for their model partition for multiple microbatches, immediately propagating them to preceding workers. By streaming multiple microbatches through the forward and backpropagation pass in parallel, the utilization of workers can be significantly increased compared to pure model parallelism, where only one batch is processed at a time. At the same time, the advantages of model parallelism are maintained, as a single worker does not need to hold the complete model. Current approaches that support pipeline parallelism are GPipe [70] and PipeDream [57, 58].

*3.2.4 Hybrid Parallelism.* Often, DL models are complex and composed of many different layers that follow a completely different architecture, which, in turn, requires different parallelization methods. Hence, hybrid approaches that mix data, model and pipeline parallelism are common.

Mesh-TensorFlow [161] is a language extension of TensorFlow that allows for combining data parallelism and model parallelism. In Mesh-TensorFlow, tensors can be split across a "mesh" of processors (such as CPUs, GPUs, or TPUs). To achieve data parallelism, data is split into shards; to achieve model parallelism, tensors are split along any of their attributes.

There are a couple of papers that propose optimizations of parallelization that are manually designed by domain experts. Krizhevsky [91] proposed to apply data parallelism for convolutional and pooling layers, as those layers are compute-heavy and only have few parameters, and model parallelism for fully connected layers, as they are light in computation, but have many parameters. In Google's Neural Machine Translation System (GNMT) [191] that powers Google Translate, they apply data parallelism, but combine it with hand-crafted model parallelism for each model replica.

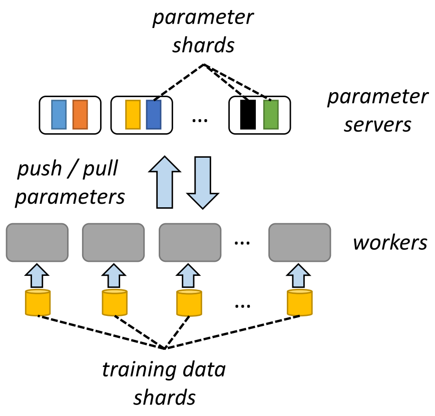

Fig. 7. Parameter server architecture.

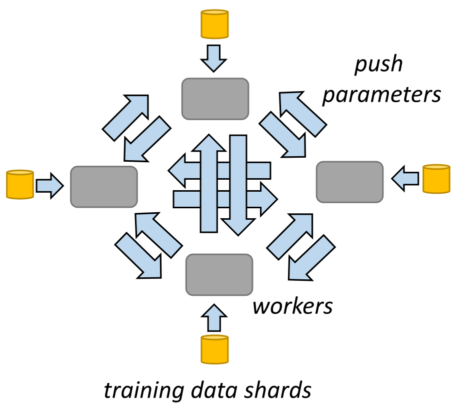

Fig. 8. All-reduce architecture.

Beyond manually designed hybrid models, recently, automated optimization approaches have been developed. Jia et al. [81] propose "layer-wise" parallelization. For each layer of a DNN, an optimal parallelization method is chosen along the tensors' dimensions at the layer. To do so, they employ a cost model and a graph search algorithm on a reduced graph that models the solution space. FlexFlow by Jia et al. [82] is an automatic parallelization optimizer that employs an execution simulator. It optimizes parallelism across four dimensions, referred to as the *SOAP space*: the sample, operation, attribute, and parameter dimension. The sample dimension refers to batches of training data and corresponds to data parallelism. The operation dimension refers to artificial neurons, the attribute dimension refers to the attributes of the tensors, and the parameter dimension refers to the weights and other model parameters. Together, the operation, attribute and parameter dimensions correspond to model parallelism [81].

## 3.3 Optimizations for Data Parallelism

Parameter synchronization in data-parallel DL systems poses three major challenges. The first challenge is *how* to synchronize the parameters. Should the workers synchronize via a centralized architecture or in a decentralized manner? The second challenge is *when* to synchronize the parameters. Should the workers be forced to synchronize after each batch, or do we allow them more freedom to work with potentially stale parameters? The third challenge is how to *minimize communication overhead* for synchronization.

*3.3.1 System Architecture.* The system architecture describes *how* the parameters of the different workers are synchronized. One of the major challenges is to provide a scalable system architecture than can deal with a large number of parallel workers that regularly update the DL model as well as receive an updated view of the model for further training. The second challenge is to keep the system easy to configure, i.e., it should be possible to yield good performance without needing extensive parameter tuning. The third challenge is to exploit lower-level primitives, e.g., communication primitives such as offered by NCCL, in an optimal way.

*(1) Centralized.* In the (logically) centralized architecture, workers periodically report their computed parameters or parameter updates to a (set of) *parameter server(s) (PSs)* (see Figure 7). Roots of the PS architecture go back to the blackboard architecture [164] and MapReduce [39], as Alex Smola reports [163]. The PS architecture is the most prominent architecture of data parallel DL systems. A common approach is to use sharding of the model parameters and distribute the shards on multiple PSs, which then can be updated in parallel [38]. Among the systems that use a

parameter server architecture are GeePS [36], DistBelief [38], TensorFlow [4], Project Adam [27], Poseidon [206], SINGA [134], SparkNet [120], and the system by Yan et al. [197].

*(2) Decentralized.* The decentralized architecture works without a PS. Instead, the workers exchange parameter updates directly via an *allreduce* operation (see Figure 8). In doing so, the *topology* of the workers plays an important role. A fully connected network, where each worker communicates with each other worker, has a communication cost that is in $O(n^2)$ with $n$ workers, so that communication becomes a bottleneck. A common alternative is to employ a ring topology (referred to as *ring-allreduce*). Horovod [160] from Uber uses NCCL to implement ring-allreduce. Baidu had one of the first proposals of using ring-allreduce for data parallel DL training [50]. The multi-GPU framework in Tencent's Mariana DL system [211] employs a similar linear topology for parameter exchange across workers. Other topologies that have been proposed are "Butterfly" [207], a tree [6], and a graph that is built based on a Halton sequence [101]. Wang et al. [183] propose a parameter sharing protocol that allows for arbitrary loop-free worker topologies that can also be dynamically changed at system run-time. The main drawback of alternative topologies, different from the fully connected topology, is that the propagation of parameter updates to all workers needs more time, as there may be multiple hops between a pair of workers.

The topology of the workers is not the only knob to reduce network load. Ako by Watcharapichat et al. [186] employs a fully connected network of workers, but partitions the gradients that are exchanged between workers (*partial gradient exchange*). In each round of synchronization, each worker only sends a single partition of the gradients to every other worker; in particular, it may send different partitions to different workers. Clearly, the communication overhead depends both on the size of a partition (which itself depends on the number of partitions) as well as on the number of workers. The number of partitions is adapted automatically in such a way that the network bandwidth remains constant independently of the number of workers.

*Comparison to centralized architecture.* The advantages of the decentralized architecture compared to the centralized one are the following, according to Li et al. [101]. By using the decentralized architecture, one avoids the need to deal with the inconveniences of implementing and tuning a parameter server. This is not only a matter of the complexity of the system code but also eases the deployment. One does not need to plan which resources to allocate for the parameter servers and for the workers. A further advantage is that fault tolerance can be achieved more easily, because there is no single point of failure such as the parameter server. When a node in the decentralized architecture fails, other nodes can easily take over its workload and the training proceeds without interruptions. Heavy-weight checkpointing of the parameter server state is not necessary.

The decentralized architecture also has disadvantages. First and foremost, communication in the decentralized architecture increases quadratically with the number of workers, if no counter-measures are taken. As discussed above, those counter-measures, such as changing the topology or partitioning the gradients, induce new complexities and trade-offs. Overall, there is no silver bullet for the problem of synchronizing parallel parameter updates.

A case study by Lian et al. [105] indicates that the decentralized architecture can, under certain conditions, perform better than the centralized architecture if the communication network is slow. However, their study is limited to synchronous parameter updates and the centralized architecture they compare to employs only a single parameter server. In such a setting, the network connecting the single central parameter server quickly becomes the bottleneck. Similar results have been reported by Iandola et al. [74] who also prefer a tree-structured allreduce architecture to a single parameter server.

Both centralized and decentralized learning are widely implemented in open-source DL frameworks. Some frameworks, such as TensorFlow and MXNet, even support both. In TensorFlow, the decentralized architecture is applied to training on a single compute node with multiple GPUs, as efficient allreduce implementations such as NCCL allreduce can be used. However, the centralized architecture is applied to multi-node training [171].

*(3) Federated.* Both the centralized and the decentralized architecture assume a controlled environment (such as a data center), a balanced and i.i.d. distribution of the training data to the workers, and a network with homogeneous, high bandwidth. In contrast to this, federated learning [90] evolves around a scenario where the training data is kept locally on users' mobile devices, and a global model is trained based on updates that the users compute on their local devices. That way, training data, which may contain privacy-sensitive information, can be completely kept locally, which can also decrease the bandwidth requirements between the mobile devices and the central data center.

The low and asynchronous bandwidth (i.e., the uplink is usually much slower than the downlink) of a mobile device's Internet connection makes it impossible to repeatedly upload the updated parameters of a large model to a centralized parameter server or to decentralized peer nodes. Konecňý et al. [90] study different forms of parameter sampling and compression to mitigate this problem. McMahan et al. [114] propose the *federated averaging* algorithm for reducing the parameter updates. Their algorithm is round-based: In each round, a fraction of the clients is selected. Each selected client computes the gradient of the loss function over all the training data that it holds. To reach convergence, it is important that the model instances on the client start from the same random initialization. Finally, a central server aggregates the gradients from the selected clients. In a comparative performance study by Nilsson et al. [130], the authors show that federated averaging is the best algorithm for federated learning, and is practically equivalent to the centralized architecture when i.i.d. training data is used. However, in the non-i.i.d. case, the centralized approach performs better than federated averaging.

Federated learning is still in an early stage and is not widely supported yet in open-source DL frameworks. Recently, first tools for federated learning were made available. TensorFlow Federated [173] is a simulator for experimenting with federated ML. PySyft [144, 153] is a Python library that enables privacy-preserving federated learning within PyTorch. In particular, PySyft applies *differential privacy* methods [5] to federated learning to prevent that sensitive information about the training data can be extracted from the model.

*3.3.2 Synchronization.* The question *when* to synchronize the parameters between the parallel workers has received a lot of attention. The main challenge in parameter synchronization is to handle the trade-off between the potential loss in training quality or convergence speed when workers perform training on a stale DL model and the synchronization cost to update the DL model on the workers. Overall, there are three different main approaches: Synchronous, bounded asynchronous, and asynchronous training. Table 1 provides an overview and categorization of the most relevant publications.

*(1) Synchronous.* In synchronous training, after each iteration (processing of a batch), the workers synchronize their parameter updates. Such a strict model can be implemented by well-known abstractions such as the Bulk Synchronous Parallel (BSP) model [175], which are in many cases already available in data analytics platforms such as Hadoop/MapReduce [39], Spark [115, 200], or Pregel [111]. The advantage of strict synchronization is that reasoning about the model convergence is easier. However, strict synchronization makes the training process prone to the *straggler problem*, where the slowest worker slows down all others [28].

GeePS [36] by Cui et al. is a parameter server implementation that is tailored to GPUs. This includes a couple of optimizations such as pre-built indexes, caching, data staging and memory management. While GeePS supports synchronous, bounded asynchronous and asynchronous

Table 1. Categorization of Approaches on Parameter Synchronization in Data-parallel Training

| | | Synchronization Model | | | System Architecture | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Ref. | Name | Sync. | Bound. Async. | Async. | Centralized | Decentralized | Federated | Year | Main Concepts |
| [149] | Hogwild | | | x | | x | | 2011 | Lock-free updates |
| [38] | Downpour SGD | | | x | x | | | 2012 | Parameter sharding, asynchronous SGD |
| [28] | Cipar et al. | | x | | x | | | 2013 | Introduces Stale Synchronous Parallel (SSP) |
| [133] | Dogwild | | | x | | x | | 2014 | Distributed Hogwild [149] |
| [35] | Cui et al. | | x | | x | | | 2014 | Applies SSP [28] |
| [102] | Li et al. | x | x | x | x | | | 2014 | Flexible consistency |
| [37] | Dai et al. | | x | | x | | | 2015 | Introduces Eager SSP |
| [101] | MALT | | | x | | x | | 2015 | Shared memory abstraction |
| [204] | Hogwild++ | | | x | | x | | 2016 | NUMA-aware Hogwild [149] |
| [36] | GeePS | x | x | x | x | | | 2016 | GPU-specialized PS |
| [83] | Jiang et al. | | x | | x | | | 2017 | Dynamic learning rates on SSP [28] |
| [184] | A-BSP | x | | | x | x | | 2018 | Aggressive synchronization |
| [89] | CROSS-BOW | x | | | | x | | 2019 | Synchronous model averaging |
| [19] | Bonawitz et al. | x | | | x | | x | 2019 | Synchronous federated learning |

parameter synchronization, it is designed to minimize the straggler problem on GPUs and, hence, achieves best convergence speed when using the synchronous approach. Wang et al. [184] propose an *aggressive synchronization* scheme that is based on BSP, named A-BSP. Different from BSP, A-BSP allows the fastest task to fetch current updates generated by the other (straggler) tasks that have only partially processed their input data. The authors have implemented A-BSP both on Spark [115, 200] as well as on the Petuum system [196]. CROSSBOW [89] by Koliousis et al. introduces *synchronous model averaging* (SMA). In SMA, data-parallel workers access a global average model to coordinate with each other. In particular, the workers independently train their model replica on their respective shard of the training data, but *correct* their model parameters according to the difference of their local models to the global average model. Bonawitz et al. [19] discuss a system design that is tailored to synchronous training for federated learning. The main challenges they address are how to deal with fluctuating device availability and churn, interrupted connectivity and limited device capabilities. To solve these challenges, they propose to employ a centralized architecture with a parameter server. The training process is divided into subsequent rounds; after each round, locally computed gradient updates are collected from the participating devices and aggregated on the parameter server using federated averaging. By selecting a new set of devices for participation in each training round, the parameter server can balance the load among devices and can flexibly react on dynamics such as device churn.

Synchronous training is implemented in a wide range of open-source DL frameworks, such as TensorFlow [4, 171] and MXNet [24, 122]. It is especially suitable for parallel training on a single, multi-GPU compute node, where communication delays are small and computational load is balanced, such that the straggler problem is not significant [123, 171].

*(2) Bounded asynchronous.* Asynchronous training makes use of the approximate nature of DL training. Recall, that DL models are mathematical functions that approximate the target function

$f^*$ as good as possible (cf. Section 2.2). Hence, small deviations and non-determinism in the training process do not necessarily harm the model accuracy. This is different from "strict" problems in data analytics, such as database queries, which are required to return a deterministic result. In bounded asynchronous training, workers may train on stale parameters, but the staleness is bounded [28]. Bounded staleness allows for a mathematical analysis and proof of the model convergence properties. The bound allows the workers for more freedom in making training progress independently from each other, which mitigates the straggler problem to some extent and increases throughput.

Cipar et al. introduced the Stale Synchronous Parallel (SSP) model [28]. Different from the BSP model, SSP allows for bounded staleness of the workers, i.e., there may be a delay between a worker updating the parameters and the effects of that update being visible to other workers. This delay is given in terms of a number of iterations. A follow-up paper by Cui et al. [35] proposes an implementation of SSP for ML jobs. Dai et al. [37] perform a theoretical analysis of SSP, comparing it against a theoretically optimal (but practically not implementable) approach. In the course of their analysis, they propose Eager SSP (ESSP), which is a novel implementation of the SSP model. In ESSP, workers eagerly pull updates from the parameter servers, as opposed to SSP where updates are only pulled when the worker state becomes too stale. ESSP is implemented in the Petuum system [196]. The parameter server by Li et al. [102] has a flexible consistency model that also supports bounded delays. Jiang et al. [83] propose to use dynamic learning rates on top of SSP to account for heterogeneous workers. Depending on a worker's speed, its learning rate is adapted such that stale updates have a less significant effect on the global parameters than fresh updates.

The bounded asynchronous model is not widely implemented in DL frameworks, as Zhang et al. [203] notice. Li [123] noted in a Github discussion that SSP was not implemented in MXNet, because the observed delays were only small due to the uniform performance of GPU-intensive operations, such that the benefits of SSP were not significant enough. There are some exceptions. The Parallel ML System (PMLS) uses Bösen [187], a bounded-asynchronous parameter server. However, PMLS and Bösen are no longer actively developed. CNTK [158] implements blockwise model update and filtering (BMUF) [23], a variant of bounded asynchronous training. Petuum, which is a commercial product, implements the bounded asynchronous model [196].

*(3) Asynchronous.* In asynchronous training, workers update their model completely independently from each other. There are no guarantees on a staleness bound, i.e., a worker may train on an arbitrarily stale model. This makes it hard to mathematically reason about the model convergence. However, it provides the workers the greatest possible flexibility in their training process, completely avoiding all straggler problems.

Hogwild [149] by Recht et al. is an asynchronous implementation of parallel SGD. The parameter update scheme of Hogwild grants the workers access to shared memory without any locks, i.e., workers can overwrite each other's updates of the model parameters. This seems dangerous due to the lost update problem: New model parameters written by one worker could directly be overwritten by another worker and, hence, would not have any effect. However, the authors show that as long as the updates of the single workers only modify small parts of the model, Hogwild achieves nearly optimal convergence. By foregoing locks, Hogwild performs by an order of magnitude faster than update schemes that lock the model parameters before each update. The Hogwild scheme has been successfully applied to the training of neural networks [42]. Dogwild [133] by Noel and Osindero is a distributed implementation of Hogwild. The authors report that using UDP congested the network stack, while using TCP did not fully utilize the communication bandwidth and also caused latency spikes, so that they use raw sockets instead. Hogwild++ [204] by Zhang et al. is an adaptation of Hogwild to NUMA-based memory architectures. Downpour SGD [38] by Dean et al. is an asynchronous SGD procedure tailored to large clusters of commodity machines. Among the main concepts of Downpour SGD are the sharded parameter server and the application

Table 2. Categorization of Approaches on Efficient Communication in Data-parallel Training

| | | Communication Optimization | | | Synchronization Model | | | System Architecture | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Ref. | Name | Precision | Compress. | Comm. Sched. | Sync. | Bound. Async. | Async. | Centralized | Decentralized | Federated | Year |
| [159] | Seide et al. | | x | | x | | | | x | | 2014 |
| [102] | Li et al. | | x | | x | x | x | x | | | 2014 |
| [55] | Gupta et al. | x | | | x | x | x | x | x | x | 2015 |
| [187] | Bösen | | | x | | x | | x | | | 2015 |
| [110] | MLNet | | | x | x | x | | x | | | 2015 |
| [209] | DoReFa-Net | x | | | x | x | x | x | x | x | 2016 |
| [8] | QSGD | | x | | x | x | x | | x | | 2017 |
| [190] | TernGrad | | x | | x | | | x | | | 2017 |
| [106] | Lin et al. | | x | | x | | | x | | x | 2018 |
| [170] | eSGD | | x | | x | | | | | x | 2018 |
| [154] | HALP | x | | | x | x | x | x | x | x | 2018 |
| [60] | TicTac | | | x | x | | | x | | | 2019 |

of adaptive learning rates [44]. Different from Hogwild, which is lock-free, Downpour SGD uses lock-guarded parameter increments. MALT [101] by Li et al. is an asynchronous ML framework that follows the decentralized architecture. It provides a shared memory abstraction for the workers that provides a scatter/gather interface as well as a higher-level vector object library.

The same as synchronous training, asynchronous training is well-established; there are many implementations in current open-source DL frameworks, such as TensorFlow [171], MXNet [122], CNTK [31], and PyTorch [145].

*3.3.3 Communication.* Synchronizing the model replicas in data-parallel training requires communication between workers and between workers and parameter servers (in the centralized architecture). The main challenge in optimizing the communication is to prevent that communication becomes the bottleneck of the overall training process, which would leave compute resources under-utilized. We identified three main approaches for communication efficiency: (1) Reducing the model precision, (2) compressing the model updates, and (3) improving the communication scheduling. The current landscape of communication approaches is categorized in Table 2.

*(1) Reducing the model precision.* Reducing the precision of the parameters of the model saves communication bandwidth when parameter updates need to be transferred over the network. Additionally, it reduces the model size, which can be useful when the model is deployed on resource-constrained hardware such as GPUs. Precision reduction can be achieved by reducing the precision of the parameters' data types, e.g., from double precision to single floating point precision or even less.

Gupta et al. [55] limited the numerical precision of DL models to 16-bit fixed-point arithmetic. They found that when applying stochastic rounding as opposed to the common round-to-nearest method, the scheme with limited precision achieves nearly the same model accuracy as when applying the traditional 32-bit floating point arithmetic that is typically used in DL. This allows for reducing the model size by half. When applied to a data-parallel DL system, this will also reduce the network bandwidth needed for communicating parameter updates between workers and parameter servers; the approach itself does not depend on a specific synchronization method or parallel architecture. DoReFa-Net [209] by Zhou et al. focuses on CNNs. Their main idea is to reduce the numerical precision of weights, activations, and gradients to different bit-widths. They report to use 1-bit weights, 2-bit activations, and 6-bit gradients on the AlexNet CNN [92] and still reach an accuracy that is competitive to a 32-bit representation. High-accuracy low-precision (HALP) by De Sa et al. [154] is an algorithm that combines two optimization techniques to reach high model accuracy despite of limited parameter precision. First, they use stochastic variance-reduced gradient (SVRG) [85] to reduce noise from gradient variance. Second, to reduce noise from parameter quantization, they introduce a new technique called *bit centering*, i.e., re-centering and re-scaling of the fixed-point representation of the parameters as the model converges. Same as Gupta et al. [55], they rely on stochastic rounding for quantization.

Model quantization is commonly applied to reduce the size of already trained models for more efficient inference, e.g., on mobile devises. Such *post-training quantization* is implemented, e.g., in TensorFlow Lite [172], MXNet [126], and PyTorch [147]. Model quantization at training time is less common; it is not widely implemented in DL frameworks.

*(2) Compressing the model updates.* The model updates communicated between workers and between workers and parameter servers can be compressed. Lossless compression is limited in the achievable compression rate, as redundancy in the parameter updates is typically limited. Instead, lossy compression is applied. The main methods in the literature are gradient quantization (reducing the number of bits per gradient) and gradient sparsification (communicating only important gradients that have a significant value).

Seide et al. [159] report on quantizing the gradients in a speech DNN to *one single bit*. To still achieve high accuracy, they propose a technique called *error-feedback*. In error-feedback, when quantizing gradients, they save the induced quantization error and add it into the respective next batch gradient before its quantization. Hence, the gradients' information is not lost by quantization, but all gradients are eventually added into the model. TernGrad [190] by Wen et al. introduces *ternary gradients*, i.e., the gradient can have the value −1, 0, or 1. To improve on the model accuracy, they propose layer-wise ternarizing (i.e., using a different quantization for each layer) and gradient clipping (i.e., limit the magnitude of each gradient before quantizing it). QSGD [8] by Alistarh et al. follows a similar approach. They apply stochastic rounding (cf. Gupta et al. [55] and De Sal et al. [154]) and statistical encoding; the key idea of the latter is that not all values are equally likely, which is exploited in the encoding scheme.

Besides quantization, another common technique is gradient sparsification. It is based on the observation that in the training process, many gradients are very small (i.e., have a value close to 0) and do not contribute much to the training. By leaving out gradients with insignificant values, the communication volume can be reduced. The parameter server by Li et al. [102] allows for gradient sparsification via user-defined filters. eSGD [170] is a gradient sparsification approach for federated architectures. Lin et al. [106] propose a gradient sparsification approach that is based on a threshold. Only gradients larger than the threshold are transmitted. The rest of the gradients are accumulated until the threshold is reached. This is similar to the error-feedback that Seide et al. [159] proposed for quantization. Lin et al. combine their sparsification approach with *momentum correction* to mitigate issues introduced by the transmission of accumulated small gradients. Further, they apply gradient clipping.

Gradient quantization and sparsification at training time are implemented in a number of open-source DL frameworks. CNTK implements the 1-bit stochastic gradient descent by Seide et al. [159]. MXNet supports 2-bit quantization with error-feedback; 1-bit quantization and sparsification techniques are on the roadmap [124].

*(3) Communication scheduling.* Communication patterns in data-parallel DL are typically *bursty*, especially in strictly synchronous systems: All workers may share their updated parameters at the same time with their peer workers or parameter servers. To prevent that the network bandwidth is exceeded and communication is delayed, the communication of the different workers can be scheduled such that it does not overlap. Furthermore, when bandwidth is constrained, but too

many parameter updates are to be sent, communication scheduling can prioritize specific messages over others, e.g., depending on freshness or on significance for the model convergence.

Bösen [187] by Wei et al. maximizes network communication efficiency by prioritizing updates that are most significant to the model convergence. TicTac [60] by Hashemi et al. is a system for communication scheduling in synchronous centralized architectures. They observe that in many ML and DL systems such as TensorFlow and PyTorch, parameters are transmitted randomly in the training and inference process. This results in high variance in iteration time, which slows down the process. To overcome that problem, TicTac enforces a schedule of network transfers that optimizes the iteration time. MLNet [110] by Mai et al. is a communication layer for centralized data-parallel ML. They combine a tree-shaped communication overlay with traffic control and prioritization to mitigate network bottlenecks.

Sophisticated communication scheduling algorithms have not found their way into open-source DL frameworks yet. This may be due to the novelty of the methods.

## 3.4 Scheduling and Elasticity

The *scheduling problem* in DL refers to how to map the (possibly parallel) DL training processes to the processing nodes in the distributed infrastructure. We identified three different aspects of scheduling in DL.

First, there is *single-tenant* scheduling (Section 3.4.1): How to map the processes (e.g., workers and parameter servers) of a single tenant, i.e., training job, to the available infrastructure? In case that mapping is dynamic, and we can change the number of training processes (e.g., number of workers and number of parameter servers) as well as the infrastructure (e.g., number of compute nodes), we also talk about *elasticity* in the scheduling problem.

Second, there is *multi-tenant* scheduling (Section 3.4.2): Given multiple competing training jobs (each having a number of processes), how to map them to the available infrastructure? The multi-tenant case introduces additional challenges such as a larger complexity and additional requirements or constraints such as fairness among the tenants.

Third, there is a specific scheduling problem that concerns the *creation* of training jobs in DL, namely, the *model architecture and hyper-parameter search* (Section 3.4.3). This problem is tightly coupled to single-tenant and multi-tenant scheduling.

### *3.4.1 Single-tenant.*

In single-tenant scheduling, we assume a dedicated, but possibly dynamic, set of resources (compute nodes, CPUs, GPUs) that is available to host a set of processes that originate from a single DL training job. With training job, we refer to all processes involved in performing the training of a single DL model. Depending on the parallelization method, this may comprise workers that train complete (data parallelism) or partial (model parallelism) model replicas as well as parameter servers. Now, scheduling needs to answer the following questions: (1) Which process is placed on which resource (such as compute node, CPU, or GPU)? (2) When or in what order are the processes that are placed on the same resource executed? (3) When and how are the number of processes and/or resources adapted?

In model parallelism, one of the major problems to be solved is to partition the model into multiple parts. We have discussed this issue and state-of-the-art approaches for addressing it in Section 3.2. Once the model is partitioned, the next important questions are *where* to place the model parts and *when* to train *which* partition of the model. As a training iteration of a model partition can only be executed when all input data of that partition is available, there are dependencies in scheduling the different model partitions. Mayer et al. [113] have formalized the scheduling problem in model-parallel DL. While they propose a couple of heuristic algorithms, none of them have been implemented in the context of DL systems. In particular, there are interdependencies between the model partition and the scheduling problem, which are yet to be fully explored. Additional challenges arise with the advent of dynamic control flow [79, 199] that renders static scheduling infeasible. Park et al. [138] propose *layer placement*, which is, however, limited to CNNs. STRADS [88] by Kim et al. is a model-parallel ML framework with an advanced scheduler. In particular, STRADS can take into account dependency structures in model partitions and is capable of prioritizing computations. To do so, the user has to implement his training task via three functions `schedule`, `update`, and `aggregate`. While the paper contains example implementations of classical ML algorithms such as LASSO and topic modeling, it is not straight-forward to implement a model-parallel DL training job via the STRADS interface.

Litz [146] by Qiao et al. is an elastic ML framework that exposes an event-driven programming model. In Litz, computations are decomposed into micro-tasks that are dynamically scheduled on a cluster. The scheduler takes into account dependencies and consistency requirements of the ML model. To enable interruption-free elasticity, the input data is "over-partitioned" across *logical executors*, which are dynamically mapped to physical resources. This allows even for transparent scaling of stateful workers, i.e., workers that keep local state that is not shared via the parameter servers or directly with peer workers. This property is useful when different model state is affected by the training of different ranges of input data, such that for faster access that portion of the model state is directly kept at the worker.

Proteus [59] by Harlap et al. exploits transient resources such as Amazon EC2 spot instances and Google Compute Engine preemptible instances. Its main concepts are a parameter server framework that is optimized for bulk addition and revocation of transient resources, and a resource allocation component that dynamically allocates transient resources to minimize the overall monetary cost per work based on highly dynamic spot markets.

CROSSBOW [89] by Koliousis et al. is a decentralized data-parallel DL system that can automatically tune the number of workers at run-time. To do so, the number of workers is increased during the training until no more increase in training throughput can be observed. This way, the available infrastructure can be utilized in an optimal way. Further, CROSSBOW comes with a dynamic task scheduler to execute workers on GPUs based on resource availability. FlexPS [71] by Huang et al. takes on the problem of *varying workloads* during the execution of ML training. As sources of varying workloads, Huang et al. mention adaptive hyper-parameters (specifically, the batch size), and advanced SGD methods such as SVRG [85]. As a result of this problem, the parallelism degree, i.e., the number of workers, needs to be adapted to re-balance the trade-off between communication and computation in data-parallel training.

### 3.4.2 *Multi-tenant.*

In a multi-tenant environment, multiple training jobs (tenants) share a common set of resources. Hence, a *resource scheduler* is responsible to schedule the processes of the different tenants on the resources. There is a large variety of general purpose resource schedulers such as Mesos [64], YARN [178], and Borg [179]. However, these are not tailored to the specific properties of DL training tasks. For instance, in DL, the convergence rate of a training task varies over time. Typically, in the beginning of training, progress is made very quickly; however, as training evolves over many epochs, the improvements on model accuracy decrease. Further, different DL training jobs may have very different training curves [205]. Taking into account these DL-specific properties allows for formulating new, DL-specific optimization goals, e.g., maximizing the overall training progress over all scheduled training jobs. Hence, new DL resource schedulers are being proposed.

Dolphin [98] by Lee et al. is an elastic centralized data-parallel ML framework. In Dolphin, the configuration of the parameter servers and workers is adapted dynamically according to a cost model and continuous monitoring. Here, the configuration refers to the number of servers and workers, the distribution of training data across workers and the distribution of model parameters across parameter servers. The system is implemented on top of Apache REEF [188], a framework for distributed applications. Optimus [141] by Peng et al. is a system that dynamically adjusts the number and placement of workers and parameter servers of a training job at run-time to achieve the best resource efficiency and training speed. To do so, it builds performance models based on sampling that estimate the number of training epochs needed until convergence and the impact of different configurations (number of workers and parameter servers) on the training speed. Then, a greedy algorithm computes the best allocation of resources to workers and parameter servers. Considering multiple concurrent training jobs to be scheduled, Optimus aims to minimize the average job completion time. An additional challenge tackled by Optimus is to divide the model parameters onto the parameter servers such that the load is balanced. Compared to the general-purpose scheduling policies Dominant Resource Fairness [49] and Tetris [52], Optimus shows significant improvements in average job completion time and makespan.[3] Jeon et al. [78] analyze log traces from a large-scale DL cluster system. In particular, they analyze the trade-off between locality constraints and queuing delays for large training jobs that occupy a lot of (GPU) resources. Further, they observe that co-locating different jobs on the same server may significantly impact their performance. Finally, they also analyze failures in DL training and the root causes why they occur. They differentiate between failures caused by the infrastructure, by the DL framework, and by the user. Based on their analysis, they propose a couple of best practices for multi-tenant DL scheduling. First, they emphasize that locality is a major design goal of schedulers that should definitely be taken into account. Second, they highlight that isolation of jobs is important to avoid performance interference. Third, they propose that new jobs should first be tested on a small dedicated set of servers before being admitted to the cluster. Ease.ml [104] is an ML service platform that employs a multi-tenant resource scheduler. Users define their training jobs in a declarative language and submit them to ease.ml via a web interface. Then, ease.ml not only schedules that job on the available resource but also automates model architecture and hyper-parameter search. The overall goal of ease.ml is to maximize the average model accuracy achieved among all tenants, i.e., users of the system. SLAQ [205] by Zhang et al. has a similar goal but supports a broader set of optimization goals. It does not only maximize average accuracy but also solves a min-max problem to provide fairness among the tenants. Ray [121, 132] from UC Berkeley is a distributed system that is specialized to support the requirements of reinforcement learning. The design of Ray makes it necessary to dynamically schedule millions of tasks per second, where each task represents a remote function invocation that may only take as little as a few milliseconds to complete. The scheduler in Ray is hierarchical with two levels: one single global scheduler and a local scheduler per node. As long as a node is not overloaded, the local scheduler schedules its tasks autonomously. However, if a local scheduler detects overload, it forwards tasks to the global scheduler, which assigns them to other nodes.

Besides publications that describe concrete multi-tenant schedulers, there are publications that describe *DL services*. IBM Fabric for Deep Learning [18] (FfDL) is a cloud-based deep-learning stack used at IBM by AI researchers. Based on FfDL, IBM offers *DL as a Service* (DLaaS) [17], a fully automated cloud solution for DL. Hauswald et al. [61] describe Djinn, an open infrastructure for DL as a service in large-scale distributed infrastructures, as well as Tonic, a suite of DL applications for image, speech, and language processing. They analyze the workloads of their system and propose a design for large-scale infrastructures that is suitable to DL workloads. One of their findings is that employing GPUs for DL training and inference can reduce total cost of ownership

[3]The makespan of a set of training jobs is the total time elapsed from the arrival of the first job to the completion of all jobs.

tremendously compared to applying only CPUs. In their analysis, they take into account upfront capital expenditures, operating costs, and financing costs. While GPUs have a higher purchase price, such investment pays off due to lower operating costs when processing DL workloads.

*3.4.3 Model Architecture and Hyper-parameter Search.* Model architecture and hyper-parameter search is a crucial problem in DL training. Given a specific task (e.g., image classification), what is the best model architecture (e.g., CNN with how many layers and what layer dimensions) that can reach the best accuracy? And what are the best hyper-parameter settings to reach model convergence quickly? Finding the answer to those questions is difficult. The typical approach is to repeatedly try out different architectures and hyper-parameter settings to find the best one, i.e., a search based on experimental evaluations [166]. The search can be random [16] or guided by more sophisticated models, such as random forests and Bayesian optimization [73] or even reinforcement learning [12, 210]. What all of those methods have in common is that they repeatedly spawn new training jobs with new configurations (architectures and hyper-parameter settings) that need to be scheduled on a shared set of distributed resources. Here, we discuss scheduling approaches that explicitly take into account workloads that are generated by such search strategies.

TuPAQ [166] by Sparks et al. is a system for automatically generating and executing model search configurations. Based on performance profiles provided by a domain expert, TuPAQ automatically optimizes the amount of resources for data parallel training. Batching together training jobs that access the same training data reduces network load and allows for further optimizations in the execution. HyperDrive [148] by Rasley et al. is a scheduler that optimizes the hyper-parameter search more aggressively than TuPAQ does. In particular, HyperDrive supports *early stopping* of the training of poorly configured jobs. Further, by incorporating the trajectory of learning curves of the trained models, HyperDrive predicts the expected accuracy improvement. Based on that, more resources are assigned to training jobs that have a high expected accuracy improvement compared to other configurations. HiveMind [127] by Narayanan et al. is a system designed to optimize the execution of multiple DL training jobs on a single GPU. The system executes a batch of models jointly and performs cross-model optimizations such as operator fusion (e.g., shared layers on different model architectures) and shared I/O (e.g., using the same training data for different configurations). Gandiva [195] by Xiao et al. is a system that schedules sets of jobs for hyper-parameter search simultaneously on a cluster of GPU-powered compute nodes. By exploiting early feedback, subsets of the jobs can be killed and resources can be freed. Based on profiling of job execution times, Gandiva employs a fine-grained application-aware time-slicing of the GPU resources to exploit them optimally. To place the jobs on GPUs, Gandiva also takes into account their memory footprint as well as communication intensity to minimize interference between job executions.

## 3.5 Data Management

One of the great challenges of large-scale DL is handling the data that is involved. On the one hand, this refers to the management of training data, whose volume easily exceeds the capabilities of a single disk or multiple disks on a single server. On the other hand, it refers to the management of the DL models, both fully trained as well as snapshots of models currently in the training phase. The training and model data need to be handled in a suitable manner, while taking into account the available distributed infrastructure, the running training processes and the resource scheduling in the data center.

*3.5.1 Training Data.* Obtaining large labeled training data sets is a hard problem. One approach to achieve this is to resort to *manual labeling*. For instance, to build the ImageNet data set, the authors relied on crowd sourcing via Amazon Mechanical Turk, which led to high accuracy of the labels [40]. However, manual labeling is expensive and time-consuming. Hence, there are several approaches to allow for training with highly noisy training data that can be easily obtained, e.g., from web image search. Xiao et al. [194] embed a label noise model into a DL framework. They train two CNNs: one of the CNNs predicts the label while the other CNNs predicts the noise type of the training data set. For training, they first pre-train both CNNs with clean training data. Then, they train the models with the noisy data, but mix in data with clean labels to prevent model drift. Overall, learning from noisy data is a vast research area (cf., e.g., References [119, 168]), which we will not cover in its entirety in this survey.

Besides obtaining data (noisy or clean), preprocessing of the training data is an important step in data management. This includes *normalization* such as cropping, resizing and other adjustments on image data [29], or data *augmentation* such as creating spectrograms from speech data [53]. Beyond normalization and augmentation, training a DL model with *distorted* training data can increase the model's robustness to noisy input data [208]. Hence, preprocessing of training data takes an important role in the overall DL architecture. For instance, Project Adam and Facebook both describe that preprocessing is performed on distinct data servers [27, 62].

Once the training data is obtained and preprocessed, it has to be provided to the training servers for feeding it into the DL models in the training iterations. Ozeri et al. [136] use simple and cheap *object storage* to store and provide the training data. The shortcoming of object storage is that the bandwidth of data provisioning is limited to about 35 MB per second for a single request, while the throughput of training data on a machine with 4 GPUs can reach up to 570 MB per second according to the authors' own measurements. They add a FUSE-based file system to the DL stack, which translates POSIX API requests into REST API requests. To overcome the read throughput limitation, their storage layer converts a single read request into multiple concurrent requests to the object storage to yield higher aggregate bandwidth. Kubernetes Volume Controller [94] (KVC) is an advanced interface for training data management on Kubernetes clusters. It provides an abstraction on training data that can be used by the training processes, and internally manages data placement and replication transparently to the user. Hoard [142] by Pinto et al. is a distributed caching system that stripes the training data across local disks of the worker machines for fast access. Training data is loaded from the backend only once and can then be provisioned from the cache for subsequent epochs and across training tasks that use the same training data (e.g., at exploratory architecture and hyper-parameter search).

*3.5.2 Model Data.* Managing the trained models is as important as the training process itself. According to Vartak et al. [177], model management involves tracking, storing and indexing of trained models. The goal of model management is to facilitate the sharing, querying and analyzing of the DL models. To make that possible, there are a number of current initiatives and approaches.

To facilitate interoperability between different DL frameworks, the Open Neural Network Exchange Format (ONNX) [3] is being developed. ONNX is the de-facto standard for exchange of model data between DL frameworks. DL frameworks that natively support ONNX are Caffe2, Chainer [7, 174], CNTK [158], MXNet [24], PyTorch [139], PaddlePaddle [137], Matlab, and SAS [155]. Moreover, model converters are available for TensorFlow [4], Keras, Apple CoreML [33], SciKit-learn [140], XGBoost [193], LIBSVM [22], and Tencent ncnn [128]. ModelDB [177] by Vartak et al. is a system for model management that provides automatic tracking of ML models, indexing, and querying via SQL or via a visual interface. Beyond the models themselves, ModelDB also manages meta data (e.g., hyper-parameters of the training process), quality metrics and training and test data sets for each model. ModelHub [116] by Miao et al. is a system that serves a similar purpose as ModelDB. Beyond providing a versioned model storage and query engine and a domain specific language for model architecture and hyper-parameter search, ModelHub also provides a repository-based model sharing system for easy exchange of DL models between different organizations.

## 4 COMPARISON OF DEEP-LEARNING FRAMEWORKS

Since the rise of DL, a large number different DL frameworks and tools have been developed and many of them are open source. They implement different concepts of parallelization and distribution, which we have discussed in Section 3. Having a large choice of open-source DL frameworks is one of the drivers of innovative DL research. In this section, we review and compare current open-source DL frameworks and tools.

### 4.1 Evaluation Criteria

We discuss and compare the frameworks according to the following criteria.

*(1) APIs.* DL frameworks should support a large range of programming languages, so that experts from different domains have easy access to them. Moreover, they should provide high-level abstractions so that a running DL use case can be created quickly without many obstacles.

*(2) Support for distribution and parallelization.* In a cloud environment, resources are available abundantly and on demand. DL frameworks should allow for easy and intuitive support for distribution and parallelization without need for custom code. We specifically examine this point with regard to the parallelization methods and optimizations we have discussed in Section 3. Here, we also discuss the possibility for users to fine-tune their deployment according to their needs. This relates to the DL frameworks' support for custom definitions of the DL model and loss functions and developing custom code for parameter servers or custom topologies in decentralized systems.

*(3) Community.* As the field of DL is dynamically evolving, with new DL model architectures and parallelization methods being proposed, it is crucial for a DL framework to have an active community that discusses and implements the most promising approaches. We measure community activity by the number of commits on the official Github repositories in the past six months (i.e., between October 2018 and March 2019) as well as the total number of topics with the respective tags on StackOverflow[4] (https://stackoverflow.com/).

We emphasize that we do not discuss and compare the performance of DL frameworks; a comprehensive performance evaluation of DL frameworks is out of the scope of this survey article. There are other studies that compare performance, e.g., by Liu et al. [108] or Jäger et al. [77].

### 4.2 Detailed Analysis

In the following, we discuss the frameworks in more detail. Table 3 provides an overview.

**Caffe** is a DL framework developed by Berkeley AI research and community contributors. It comes with command line, Python and Matlab APIs. A specialty of Caffe is the *model zoo*, a collection of pre-trained models for an easy start. It runs on CUDA platforms (using the cuDNN library) for easy parallelization on GPUs. Caffe does not support distributed training out-of-the-box. However, there are forks and extensions of Caffe such as Intel Caffe[5] and CaffeOnSpark[6] that support distributed training. There is only little information available in the Caffe documentation of how to customize the framework, e.g., to develop new loss functions. As Caffe does not support multi-node deployment, custom parallelization techniques can not be implemented either.

[4]Due to limitations of the StackOverflow search, we did not confine the search to recent topics, but we report the overall numbers without time constraint.
[5]https://github.com/intel/caffe.
[6]https://github.com/yahoo/CaffeOnSpark.

Table 3. Comparison of Open-source DL Frameworks and Libraries

| Name | Papers | API | Distribution and Parallelization | Community |
|---|---|---|---|---|
| Caffe | [80] | CLI, Python, Matlab | No native support for distribution. | Github: 2<br>StOv: 2,750 |
| Caffe2 | n/a | C++, Python | • Decentralized only<br>• Synchronous only<br>• Model quantization supported<br>• Gradient quantization not supported<br>• Communication scheduling not supported | Github: n/a<br>StOv: 116 |
| Chainer | [7, 174] | Python | • Decentralized only<br>• Synchronous only<br>• Model quantization not supported<br>• Gradient quantization not supported<br>• Communication scheduling not supported | Github: 3,939<br>StOv: 132 |
| CNTK | [158] | C++, C#, Python, Brain-Script | • Centralized and decentralized<br>• Bounded asynchronous training via BMUF [23]<br>• Model quantization not supported<br>• 1-bit gradient quantization [159] supported<br>• Communication scheduling not supported | Github: 138<br>StOv: 488 |
| DL4j | n/a | Java | • Centralized and decentralized<br>• Synchronous and asynchronous<br>• Model quantization not supported<br>• Modified 1-bit gradient quantization by Strom [43, 167] supported<br>• Communication scheduling not supported | Github: 390<br>StOv: 243 |
| Keras | n/a | CNTK, DL4j, TensorFlow, Theano | • Model quantization supported<br>• Higher-level concepts must be implemented in the DL framework that employs Keras | Github: 310<br>StOv: 14,630 |
| MXNet | [24] | C++, Go, Java-Script, Julia, Matlab, Perl, Python, R, Scala, Wolfram | • Centralized only<br>• Synchronous and asynchronous<br>• Model quantization supported<br>• 2-bit gradient quantization with error-feedback supported [124]<br>• Communication scheduling not supported | Github: 837<br>StOv: 455 |
| PyTorch | [139] | C++, Python | • Centralized and decentralized<br>• Synchronous and asynchronous<br>• Model quantization not supported<br>• Gradient quantization not supported<br>• Communication scheduling not supported | Github: 3,484<br>StOv: 2,413 |
| SINGA | [134] | C++, Python | • Centralized and decentralized<br>• Synchronous and asynchronous<br>• Model quantization not supported<br>• Gradient quantization not supported<br>• Communication scheduling not supported | Github: 44<br>StOv: 0 |
| TensorFlow | [4] | C++, Go, Java, Java-Script, Python, Swift | • Centralized<br>• Synchronous and asynchronous<br>• Model quantization supported<br>• Gradient quantization not supported<br>• Communication scheduling not supported | Github: 10,930<br>StOv: 39,334 |
| Theano | [15] | Python | No native support for distribution. | Github: 55<br>StOv: 2,389 |

StOv: StackOverflow.

Commit activity on Github has almost completely ceased. On StackOverflow, there are 2,750 questions tagged with “Caffe,” a high value compared to other frameworks.

**Caffe2** is a successor of the Caffe framework developed by Facebook and community contributors. The API is available in C++ and Python. The models from Caffe can be easily converted to work with Caffe2. Beyond that, Caffe2 provides its own model zoo as well. Caffe2 extends Caffe in the following points. First, Caffe2 naturally supports distributed training. There is native support for decentralized data-parallel training using the synchronous model; there is no support for (bounded) asynchronous training and no parameter server architecture. There is also native support for quantized models, i.e., models with reduced data type precision. Recently, the code of Caffe2 has been merged into PyTorch. This makes it hard to assess the update frequency of the Caffe2 code. On StackOverflow, there are 116 questions tagged with "Caffe2," a rather low value compared to other frameworks.

**Chainer** is a DL framework developed by the Japanese company Preferred Networks with several industrial partners and community contributors. It is written in Python and only has a Python interface. There is good documentation on how to write custom functions, optimizer, and trainers. *ChainerMN* is an extension package that enables distributed and parallel DL on multiple nodes. It supports data parallelism via a decentralized all-reduce architecture using the synchronous training method (no parameter server or asynchronous training are supported). There were 3,939 commits to the official Github repository in the past six months, which is a comparably high value. On StackOverflow, there are 132 questions tagged with "Chainer," a rather low value compared to other frameworks.

**CNTK** (Microsoft Cognitive Toolkit) is a DL framework developed by Microsoft and community contributors. The API is available in C++, C# and Python. Additionally, CNTK provides a custom model description language called BrainScript. The model evaluation function can also be used from Java programs. Data-parallel and distributed training is supported out-of-the-box. The 1-bit stochastic gradient descent by Seide et al. [159] is integrated into the framework. CNTK supports the centralized architecture with parameter servers, using asynchronous training or blockwise model update and filtering (BMUF) [23], a variant of bounded asynchronous training. Currently, model parallelism is not supported by CNTK. Extending CNTK is easy. New operators, loss functions, and so on, can be implemented with an API. There were 138 commits to the official Github repository in the past six months, which is a comparably low value. On StackOverflow, there are 488 questions tagged with "CNTK," an average value compared to other frameworks.

**Deeplearning4j** is a DL framework developed by the company Skymind and community contributors organized in the Eclipse foundation. The framework is written in Java and C++ (for core components), and the API is available in Java, which makes it accessible for Java, Scala and Clojure projects (but not from Python). It supports distributed and parallel training by using Spark. There are two variants of data-parallel training implemented. First, a decentralized asynchronous approach proposed by Strom [167] that also incorporates quantization of gradients. Second, centralized synchronous training with a single parameter server. There is no support for model parallelism. It is easily possible to create custom layer implementations, but more sophisticated customization (loss functions, parallelization configurations, etc.) is not supported. There were 390 commits to the official Github repository in the past six months, which is an average value. On StackOverflow, there are 243 questions tagged with "Deeplearning4j," a rather low value compared to other frameworks.

**Keras** is not a DL framework, but a DL library that can be integrated into many other DL frameworks, such as CNTK, Deeplearning4j, TensorFlow, and Theano. It is developed as a community project, initiated by F. Chollet. Keras is written in Python, which allows for its easy integration into other Python-based frameworks. Parallel training on GPUs is naturally supported; higher-level parallelization concepts must be implemented by the DL framework that uses Keras. Model quantization (to 8-bit model weights) is supported directly in Keras. The library is easily extensible with new modules. There were 310 commits to the official Github repository in the past six months, which is an average value. On StackOverflow, there are 14,630 questions tagged with "Keras," a very high value compared to other frameworks.

**MXNet** is a DL framework and an Apache project (incubating). Its API is available for C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl, and Wolfram Language. MXNet supports a wide range of parallelization approaches. Model parallelism is supported for multiple GPUs on a single node; there is no support for multi-node model parallelism though. Data parallelism is realized via the centralized architecture with support for using multiple parameter servers via a sharded key-value store. Both synchronous and asynchronous training are supported out-of-the-box. MXNet also supports post-training 8-bit model quantization tailored to the Intel(R) Math Kernel Library for Deep Neural Networks (Intel(R) MKL-DNN) [126]. In the training process, 2-bit gradient quantization with error-feedback is supported [124]. It is easy to implement custom operators or layers as well as loss functions. There were 837 commits to the official Github repository in the past six months, which is an average value. On StackOverflow, there are 455 questions tagged with "MXNet," an average value compared to other frameworks.

**PyTorch** is a DL framework developed by Facebook and community contributors. Its API is available for C++ and Python. PyTorch has native support for distributed, data-parallel training, as well as model-parallel training. For data-parallel training, PyTorch implements the decentralized architecture and supports synchronous as well as asynchronous training. PyTorch supports model quantization via the QNNPACK library [147]. Gradient quantization is not supported out-of-the-box. Writing new operators or layers is easily done via extending an interface; it is also possible to write custom loss functions. There were 3,484 commits to the official Github repository in the past six months, which is a comparably high value. On StackOverflow, there are 2,413 questions tagged with "PyTorch," a rather high value compared to other frameworks.

**SINGA** is a DL framework and Apache project (incubating) that is developed by community contributors. The initiators of the project are from the National University of Singapore. It has APIs in C++ and Python. Singa has native support for distributed, data-parallel and model-parallel training, as well as hybrid parallelism (combining data and model parallelism). Data parallelism is implemented via the centralized approach with support for multiple parameter servers. However, the decentralized architecture can be emulated by employing each worker with a local parameter server. Both synchronous and asynchronous training are supported. There is no support for model or gradient quantization. Customization is more difficult than in the other frameworks: The documentation does not contain any hints on how to implement custom layers or loss functions. There were 44 commits to the official Github repository in the past six months, which is a comparably low value. On StackOverflow, there are no questions tagged with "Singa" or "Apache Singa," and only one single question is returned when searching for the keyword "Singa."

**TensorFlow** is an ML framework developed by Google and community contributors. The API is available for C++, Go, Java, JavaScript, Python, and Swift. Additionally, the community offers bindings for C#, Haskell, Ruby, Rust, and Scala. TensorFlow natively supports distributed and parallel training. In particular, it supports both model parallelism and data parallelism. In data parallelism, the centralized approach via parameter servers is supported, using either asynchronous or synchronous training. Trained models can be quantized using TensorFlow Lite [172]. Currently, there is no native support for gradient quantization or communication scheduling. Customization of layers and loss functions is straight forward via implementing the available interfaces. There were 10,930 commits to the official Github repository in the past six months, which is an extremely high value. On StackOverflow, there are 39,334 questions tagged with "TensorFlow," which is the highest number among all analyzed DL frameworks.

**Theano** is a DL framework developed by Montreal Institute for Learning Algorithms at the Université de Montréal. The API is available only for Python. There is no support for distributed training on multiple nodes. However, using multiple GPUs on a single node is supported. Theano supports model parallelism, but no data parallelism. New layers can be implemented via an interface.

It is also possible to define custom loss functions. At the time of writing this survey, commits to the official Github repository have a low frequency. According to a posting on the Theano mailing list[7], major development of Theano ceased with the release of version 1.0.; however, new maintenance releases have been issues since then. There were still 55 commits to the official Github repository in the past six months. On StackOverflow, there are 2,389 questions tagged with "Theano," a rather high value compared to other frameworks.

**Others.** There are a couple of other frameworks that we do not cover in detail in our comparison for various reasons. Minerva [182] is an open-sourced DL system, but has not been maintained for the past 4 years. SparkNet [120] allows for distributed DL on Spark, but has not been maintained for the past 3 years. Neon [129] is another DL framework that has ceased development for more than 1 year. Scikit-learn [140] is an ML framework and it is not specific to DL. While neural network training is implemented, there is no support for using GPUs or distributed training. The Weka workbench [48] is a collection of ML and data mining algorithms. WekaDeeplearning4j [189] is a DL package for the Weka workbench. As backend, it uses Deeplearning4j, which we have discussed above.

## 5 CONCLUSIONS AND OUTLOOK

DL is becoming increasingly important in industry and academia and is without doubt one of the most impactful revolutions in computer science in the past years. However, the rapid pace in which the field is developing makes it difficult to keep an overview. In particular, DL is currently investigated from many different perspectives and in different communities. In this survey, we took a deeper look into DL from the perspective of *scalable distributed systems*. We investigated the main challenges to make DL systems scale, and have reviewed the common techniques that have been proposed by researchers to tackle those challenges. This included an analysis of the distributed infrastructures used in DL training as well as techniques for parallelization, scheduling and data management. Finally, we provided an overview and comparison of the current open-sourced DL systems and tools, and analyzed which of the techniques developed in research have actually been implemented. We saw that the wide range of techniques for scalable DL are implemented in open-source DL frameworks. This shows that there is a fruitful interaction between research and practical applications, which is one of the reasons why DL has gained such a large momentum.

We can draw from our survey a couple of insights on how to design future DL infrastructures and tools. In our opinion, management of training and model data becomes a larger challenge with the proliferation of more training data and more DL models. This demands better tool support such that new bottlenecks and limitations for DL scalability can be mitigated. Furthermore, current developments and advances in decentralized training, e.g., federated learning, may change the requirements and design of DL infrastructures and tools. If the infrastructure becomes more heterogeneous, then this must be reflected in DL tools that can not only just deal with such heterogeneity, but even exploit it to optimize the training process.

Looking into the future, we see a couple of trends that will be important in the next years. While research on scalable DL was mostly focused on the parallelization and distribution aspects of DL training, there is a need to investigate other parts of the DL environment, such as data management and multi-tenant scheduling. This is a large field for research in the distributed systems and database community. Furthermore, DL serving, i.e., using trained DL models for inference, receives growing attention [34, 54, 76]. Although DL serving is closely related to DL training, the requirements and, hence, the solutions are totally different. Another important aspect of DL is

[7]https://groups.google.com/forum/#!topic/theano-users/7Poq8BZutbY.

privacy [5, 103, 162], which receives growing attention due to an increasing awareness in the society for privacy issues in the era of Big Data, fueled by legislative reforms such as the General Data Protection Regulation (GDPR) in the European Union. There is an interesting trade-off between the ever-increasing demand for more training data to improve DL models and the principle of data avoidance and data economy to protect privacy.

## REFERENCES

[1] NVIDIA. NVIDIA Collective Communications Library (NCCL). Retrieved from https://developer.nvidia.com/nccl.

[2] NVIDIA. NVIDIA DGX Station. Retrieved from https://www.nvidia.com/en-us/data-center/dgx-station/.

[3] ONNX Project Contributors. ONNX. Retrieved from https://onnx.ai/.

[4] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Savannah, GA, 265–283. Retrieved from https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[5] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. ACM, New York, NY, 308–318. DOI: https://doi.org/10.1145/2976749.2978318

[6] Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. 2014. A reliable effective terascale linear learning system. *J. Mach. Learn. Res.* 15 (2014), 1111–1133. Retrieved from http://jmlr.org/papers/v15/agarwal14a.html.

[7] Takuya Akiba, Keisuke Fukuda, and Shuji Suzuki. 2017. ChainerMN: Scalable distributed deep learning framework. In *Proceedings of the Workshop on ML Systems in the 31st Annual Conference on Neural Information Processing Systems (NIPS'17)*. Retrieved from http://learningsys.org/nips17/assets/papers/paper_25.pdf.

[8] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 1709–1720. Retrieved from http://papers.nips.cc/paper/6768-qsgd-communication-efficient-sgd-via-gradient-quantization-and-encoding.pdf.

[9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A view of cloud computing. *Commun. ACM* 53, 4 (Apr. 2010), 50–58. DOI: https://doi.org/10.1145/1721654.1721672

[10] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Process. Mag.* 34, 6 (Nov. 2017), 26–38. DOI: https://doi.org/10.1109/MSP.2017.2743240

[11] E. Azarkhish, D. Rossi, I. Loi, and L. Benini. 2018. Neurostream: Scalable and energy efficient deep learning with smart memory cubes. *IEEE Trans. Parallel Distrib. Syst.* 29, 2 (Feb. 2018), 420–434. DOI: https://doi.org/10.1109/TPDS.2017.2752706

[12] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing neural network architectures using reinforcement learning. In *Proceedings of the International Conference on Learning Representations*.

[13] Luiz Andre Barroso and Urs Hoelzle. 2009. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines* (1st ed.). Morgan and Claypool Publishers.

[14] Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. Retrieved from http://arxiv.org/abs/1802.09941.

[15] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. 2011. Theano: Deep learning on GPUs with python. In *Proceedings of the BigLearning Workshop (NIPS'11)*, Vol. 3. Citeseer, 1–48.

[16] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, 1 (Feb. 2012), 281–305. Retrieved from http://dl.acm.org/citation.cfm?id=2503308.2188395.

[17] B. Bhattacharjee, S. Boag, C. Doshi, P. Dube, B. Herta, V. Ishakian, K. R. Jayaram, R. Khalaf, A. Krishna, Y. B. Li, V. Muthusamy, R. Puri, Y. Ren, F. Rosenberg, S. R. Seelam, Y. Wang, J. M. Zhang, and L. Zhang. 2017. IBM deep learning service. *IBM J. Res. Dev.* 61, 4/5 (July 2017), 10:1–10:11. DOI: https://doi.org/10.1147/JRD.2017.2716578

[18] Scott Boag, Parijat Dube, Benjamin Herta, Waldemar Hummer, Vatche Ishakian, K. R. Jayaram, Michael Kalantar, Vinod Muthusamy, Priya Nagpurkar, and Florian Rosenberg. 2017. Scalable multi-framework multi-tenant lifecycle management of deep learning training jobs. In *Proceedings of the Workshop on ML Systems (NIPS'17)*. Retrieved from http://hummer.io/docs/2017-nips-ffdl.pdf.

[19] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe M. Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards federated learning at scale: System design. In *Proceedings of the Conference on Systems and Machine Learning (SysML'19)*. Retrieved from https://arxiv.org/abs/1902.01046.

[20] Fedor Borisyuk, Albert Gordo, and Viswanath Sivakumar. 2018. Rosetta: Large scale system for text detection and recognition in images. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. ACM, New York, NY, 71–79. DOI:https://doi.org/10.1145/3219819.3219861

[21] Irem Boybat, Manuel Le Gallo, S. R. Nandakumar, Timoleon Moraitis, Thomas Parnell, Tomas Tuma, Bipin Rajendran, Yusuf Leblebici, Abu Sebastian, and Evangelos Eleftheriou. 2018. Neuromorphic computing with multi-memristive synapses. *Nature Commun.* 9, 1 (2018), 2514.

[22] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2, 3 (2011), 27:1–27:27. Retrieved from http://www.csie.ntu.edu.tw/cjlin/libsvm.

[23] K. Chen and Q. Huo. 2016. Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'16)*. 5880–5884. DOI:https://doi.org/10.1109/ICASSP.2016.7472805

[24] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. In *Proceedings of Workshop on Machine Learning Systems (LearningSys'15) in the 29th Annual Conference on Neural Information Processing Systems (NIPS'15)*. Retrieved from http://learningsys.org/papers/LearningSys_2015_paper_1.pdf.

[25] X. Chen and X. Lin. 2014. Big data deep learning: Challenges and perspectives. *IEEE Access* 2 (2014), 514–525. DOI:https://doi.org/10.1109/ACCESS.2014.2325029

[26] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient primitives for deep learning. Retrieved from http://arxiv.org/abs/1410.0759.

[27] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project adam: Building an efficient and scalable deep learning training system. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 571–582. Retrieved from https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi.

[28] James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R. Ganger, Garth Gibson, Kimberly Keeton, and Eric Xing. 2013. Solving the straggler problem with bounded staleness. In *Proceedings of the 14th Workshop on Hot Topics in Operating Systems (HotOS'13)*. USENIX, Santa Ana Pueblo, NM. Retrieved from https://www.usenix.org/conference/hotos13/solving-straggler-problem-bounded-staleness.

[29] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. 2012. Multi-column deep neural network for traffic sign classification. *Neural Netw.* 32 (2012), 333–338.

[30] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. 2010. Deep, big, simple neural nets for handwritten digit recognition. *Neural Comput.* 22, 12 (2010), 3207–3220.

[31] Microsoft. Multiple GPUs and Machines – Cognitive Toolkit – CNTK | Microsoft Docs. Retrieved from https://docs.microsoft.com/en-us/cognitive-toolkit/multiple-gpus-and-machines.

[32] Adam Coates, Brody Huval, Tao Wang, David J. Wu, Andrew Y. Ng, and Bryan Catanzaro. 2013. Deep learning with COTS HPC systems. In *Proceedings of the 30th International Conference on International Conference on Machine Learning Volume 28 (ICML'13)*. JMLR.org, III–1337–III–1345. Retrieved from http://dl.acm.org/citation.cfm?id=3042817.3043086.

[33] Apple. Apple CoreML. Retrieved from https://developer.apple.com/machine-learning/.

[34] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A low-latency online prediction serving system. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*. USENIX Association, Boston, MA, 613–627. Retrieved from https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw.

[35] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. 2014. Exploiting bounded staleness to speed up big data analytics. In *Proceedings of the USENIX Conference on USENIX Annual Technical Conference (USENIX ATC'14)*. USENIX Association, Berkeley, CA, 37–48. Retrieved from http://dl.acm.org/citation.cfm?id=2643634.2643639.

[36] Henggang Cui, Hao Zhang, Gregory R. Ganger, Phillip B. Gibbons, and Eric P. Xing. 2016. GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys'16)*. ACM, New York, NY. DOI:https://doi.org/10.1145/2901318.2901323

[37] Wei Dai, Abhimanu Kumar, Jinliang Wei, Qirong Ho, Garth Gibson, and Eric P. Xing. 2015. High-performance distributed ML at scale through parameter server consistency models. In *Proceedings of the 29th AAAI Conference*

*on Artificial Intelligence (AAAI'15)*. AAAI Press, 79–87. Retrieved from http://dl.acm.org/citation.cfm?id=2887007.2887019.

[38] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*. MIT Press, 1223–1231.

[39] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. DOI: https://doi.org/10.1145/1327452.1327492

[40] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. DOI: https://doi.org/10.1109/CVPR.2009.5206848

[41] Li Deng. 2014. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Trans. Signal Info. Process.* 3 (2014), e2. DOI: https://doi.org/10.1017/atsip.2013.9

[42] Valentin Deyringer, Alexander Fraser, Helmut Schmid, and Tsuyoshi Okita. 2017. Parallelization of neural network training for NLP with Hogwild! *Prague Bull. Math. Linguist.* 109, 1 (2017), 29–38. Retrieved from https://content.sciendo.com/view/journals/pralin/109/1/article-p29.xml.

[43] Eclipse Deeplearning4j. Deeplearning4j on Spark: Technical Explanation. Retrieved from https://deeplearning4j.org/docs/latest/deeplearning4j-scaleout-technicalref.

[44] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* 12 (July 2011), 2121–2159. Retrieved from http://dl.acm.org/citation.cfm?id=1953048.2021068.

[45] Bradley J. Erickson, Panagiotis Korfiatis, Zeynettin Akkus, Timothy Kline, and Kenneth Philbrick. 2017. Toolkits and libraries for deep learning. *J. Dig. Imag.* 30, 4 (Aug. 2017), 400–405. DOI: https://doi.org/10.1007/s10278-017-9965-6

[46] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. 2011. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*. 109–116. DOI: https://doi.org/10.1109/CVPRW.2011.5981829

[47] Volker Fischer, Jan Koehler, and Thomas Pfeil. 2018. The streaming rollout of deep networks—Toward fully model-parallel execution. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 4043–4054. Retrieved from http://papers.nips.cc/paper/7659-the-streaming-rollout-of-deep-networks-towards-fully-model-parallel-execution.pdf.

[48] Eibe Frank, Mark Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H. Witten, and Len Trigg. 2010. *Weka-A Machine Learning Workbench for Data Mining*. Springer US, Boston, MA, 1269–1277. DOI: https://doi.org/10.1007/978-0-387-09823-4_66

[49] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*. USENIX Association, Berkeley, CA, 323–336. http://dl.acm.org/citation.cfm?id=1972457.1972490

[50] Andrew Gibiansky. [n.d.]. Bringing HPC Techniques to Deep Learning. Retrieved from http://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/. http://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/.

[51] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems*. MIT Press, 2672–2680.

[52] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. In *Proceedings of the ACM Conference on SIGCOMM (SIGCOMM'14)*. ACM, New York, NY, 455–466. DOI: https://doi.org/10.1145/2619239.2626334

[53] Alex Graves and Navdeep Jaitly. 2014. Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*. MIT Press, 1764–1772.

[54] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S. McKinley, and Björn B. Brandenburg. 2017. Swayam: Distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference (Middleware'17)*. ACM, New York, NY, 109–120. DOI: https://doi.org/10.1145/3135974.3135993

[55] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning Volume 37 (ICML'15)*. JMLR.org, 1737–1746. Retrieved from http://dl.acm.org/citation.cfm?id=3045118.3045303.

[56] A. Halevy, P. Norvig, and F. Pereira. 2009. The unreasonable effectiveness of data. *IEEE Intell. Syst.* 24, 2 (March 2009), 8–12. DOI: https://doi.org/10.1109/MIS.2009.36

[57] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, and Phillip B. Gibbons. 2018. PipeDream: Fast and efficient pipeline parallel DNN training. Retrieved from http://arxiv.org/abs/1806.03377.

[58] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Gregory R. Ganger, and Phillip B. Gibbons. 2018. PipeDream: Pipeline parallelism for DNN training. In *Proceedings of the Conference on Systems and Machine Learning (SysML'18)*.

[59] Aaron Harlap, Alexey Tumanov, Andrew Chung, Gregory R. Ganger, and Phillip B. Gibbons. 2017. Proteus: Agile ML elasticity through tiered reliability in dynamic resource markets. In *Proceedings of the 12th European Conference on Computer Systems (EuroSys'17)*. ACM, New York, NY, 589–604. DOI: https://doi.org/10.1145/3064176.3064182

[60] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy H. Campbell. 2019. TicTac: Accelerating distributed deep learning with communication scheduling. In *Proceedings of the Conference on Systems and Machine Learning (SysML'19)*.

[61] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang. 2015. DjiNN and tonic: DNN as a service and its implications for future warehouse scale computers. In *Proceedings of the ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. 27–40. DOI: https://doi.org/10.1145/2749469.2749472

[62] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang. 2018. Applied machine learning at Facebook: A datacenter infrastructure perspective. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'18)*. 620–629. DOI: https://doi.org/10.1109/HPCA.2018.00059

[63] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory F. Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. 2017. Deep learning scaling is predictable, empirically. Retrieved from http://arxiv.org/abs/1712.00409.

[64] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*. USENIX Association, Berkeley, CA, 295–308. Retrieved from http://dl.acm.org/citation.cfm?id=1972457.1972488.

[65] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.* 29, 6 (Nov. 2012), 82–97. DOI: https://doi.org/10.1109/MSP.2012.2205597

[66] G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507. DOI: https://doi.org/10.1126/science.1127647 arXiv:http://science.sciencemag.org/content/313/5786/504.full.pdf

[67] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9, 8 (1997), 1735–1780. DOI: https://doi.org/10.1162/neco.1997.9.8.1735.

[68] M. D. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. 2019. A comprehensive survey of deep learning for image captioning. *ACM Comput. Surv.* 51, 6 (Feb. 2019). DOI: https://doi.org/10.1145/3295748

[69] Xuedong Huang, James Baker, and Raj Reddy. 2014. A historical perspective of speech recognition. *Commun. ACM* 57, 1 (Jan. 2014), 94–103. DOI: https://doi.org/10.1145/2500887

[70] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. 2018. GPipe: Efficient training of giant neural networks using pipeline parallelism. Retrieved from http://arxiv.org/abs/1811.06965.

[71] Yuzhen Huang, Tatiana Jin, Yidi Wu, Zhenkun Cai, Xiao Yan, Fan Yang, Jinfeng Li, Yuying Guo, and James Cheng. 2018. FlexPS: Flexible parallelism control in parameter server architecture. *Proc. VLDB Endow.* 11, 5 (Jan. 2018), 566–579. DOI: https://doi.org/10.1145/3187009.3177734

[72] David H. Hubel and Torsten N. Wiesel. 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* 160, 1 (1962), 106–154.

[73] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2014. An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31st International Conference on International Conference on Machine Learning Volume 32 (ICML'14)*. JMLR.org, I–754–I–762. http://dl.acm.org/citation.cfm?id=3044805.3044891

[74] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, and Kurt Keutzer. 2016. FireCaffe: Near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*.

[75] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. 2016. Let there be color!: Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Trans. Graph.* 35, 4, Article 110 (July 2016), 11 pages. DOI: https://doi.org/10.1145/2897824.2925974

[76] V. Ishakian, V. Muthusamy, and A. Slominski. 2018. Serving deep learning models in a serverless platform. In *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E'18)*. 257–262. DOI: https://doi.org/10.1109/IC2E.2018.00052

[77] Sebastian Jäger, Hans-Peter Zorn, Stefan Igel, and Christian Zirpins. 2018. Parallelized training of deep NN: Comparison of current concepts and frameworks. In *Proceedings of the 2nd Workshop on Distributed Infrastructures for Deep Learning (DIDL'18).* ACM, New York, NY, 15–20. DOI: https://doi.org/10.1145/3286490.3286561

[78] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. *arXiv preprint arXiv:1901.05758.*

[79] Eunji Jeong, Joo Seong Jeong, Soojeong Kim, Gyeong-In Yu, and Byung-Gon Chun. 2018. Improving the expressiveness of deep learning frameworks with recursion. In *Proceedings of the 13th EuroSys Conference (EuroSys'18).* ACM, New York, NY. DOI: https://doi.org/10.1145/3190508.3190530

[80] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. Retrieved from http://arxiv.org/abs/1408.5093.

[81] Zhihao Jia, Sina Lin, Charles R. Qi, and Alex Aiken. 2018. Exploring hidden dimensions in accelerating convolutional neural networks. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR, StockholmsmÃ¤ssan, Stockholm Sweden, 2274–2283. Retrieved from http://proceedings.mlr.press/v80/jia18a.html.

[82] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2018. Beyond data and model parallelism for deep neural networks. *CoRR* abs/1807.05358 (2018). Retrieved from http://arxiv.org/abs/1807.05358.

[83] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware distributed parameter servers. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'17).* ACM, New York, NY, 463–478. DOI: https://doi.org/10.1145/3035918.3035933

[84] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Trans. Assoc. Comput. Linguist.* 5, 1 (2017), 339–351.

[85] Rie Johnson and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Proceedings of the 26th International Conference on Neural Information Processing Systems Volume 1 (NIPS'13).* Curran Associates Inc., USA, 315–323. Retrieved from http://dl.acm.org/citation.cfm?id=2999611.2999647.

[86] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA'17).* 1–12. DOI: https://doi.org/10.1145/3079856.3080246

[87] Peter Karkus, David Hsu, and Wee Sun Lee. 2017. QMDP-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4694–4704. Retrieved from http://papers.nips.cc/paper/7055-qmdp-net-deep-learning-for-planning-under-partial-observability.pdf.

[88] Jin Kyu Kim, Qirong Ho, Seunghak Lee, Xun Zheng, Wei Dai, Garth A. Gibson, and Eric P. Xing. 2016. STRADS: A distributed framework for scheduled model parallel machine learning. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys'16).* ACM, New York, NY. DOI: https://doi.org/10.1145/2901318.2901331

[89] Alexandros Koliousis, Pijika Watcharapichat, Matthias Weidlich, Luo Mai, Paolo Costa, and Peter R. Pietzuch. 2019. CROSSBOW: Scaling deep learning with small batch sizes on multi-GPU servers. Retrieved from http://arxiv.org/abs/1901.02244.

[90] Jakub Konecňý, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. In *Proceedings of the NIPS Workshop on Private Multi-Party Machine Learning.* Retrieved from https://arxiv.org/abs/1610.05492.

[91] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. Retrieved from http://arxiv.org/abs/1404.5997.

[92] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems Volume 1 (NIPS'12).* Curran Associates Inc., 1097–1105. Retrieved from http://dl.acm.org/citation.cfm?id=2999134.2999257.

[93] Arun Kumar, Matthias Boehm, and Jun Yang. 2017. Data management in machine learning: Challenges, techniques, and systems. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'17).* ACM, New York, NY, 1717–1722. DOI: https://doi.org/10.1145/3035918.3054775

[94] Intel. Data Management Tailored for Machine Learning Workloads in Kubernetes. Retrieved from https://www.intel.ai/kubernetes-volume-controller-kvc-data-management-tailored-for-machine-learning-workloads-in-kubernetes.

[95] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.

[96] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov. 1998), 2278–2324. DOI: https://doi.org/10.1109/5.726791

[97] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. 2010. Debunking the 100X GPU vs. CPU Myth: An evaluation of throughput computing on CPU and GPU. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. ACM, New York, NY, 451–460. DOI: https://doi.org/10.1145/1815961.1816021

[98] Yun Seong Lee Lee, Markus Weimer, Youngseok Yang, and Gyeong-In Yu. 2016. Dolphin: Runtime optimization for distributed machine learning. In *Proceedings of International Conference on Machine Learning (ICML'16)*.

[99] Ian Lenz, Honglak Lee, and Ashutosh Saxena. 2015. Deep learning for detecting robotic grasps. *Int. J. Robot. Res.* 34, 4 5 (2015), 705–724. DOI: https://doi.org/10.1177/0278364914549607

[100] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. 2019. Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. Retrieved from http://arxiv.org/abs/1903.04611.

[101] Hao Li, Asim Kadav, Erik Kruus, and Cristian Ungureanu. 2015. MALT: Distributed data-parallelism for existing ML applications. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys'15)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2741948.2741965

[102] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Broomfield, CO, 583–598. Retrieved from https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu.

[103] Ping Li, Jin Li, Zhengan Huang, Tong Li, Chong-Zhi Gao, Siu-Ming Yiu, and Kai Chen. 2017. Multi-key privacy-preserving deep learning in cloud computing. *Future Gen. Comput. Syst.* 74 (2017), 76–85. DOI: https://doi.org/10.1016/j.future.2017.02.006

[104] Tian Li, Jie Zhong, Ji Liu, Wentao Wu, and Ce Zhang. 2018. Ease.Ml: Towards multi-tenant resource sharing for machine learning workloads. *Proc. VLDB Endow.* 11, 5 (Jan. 2018), 607–620. DOI: https://doi.org/10.1145/3187009.3177737

[105] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*. MIT Press, 5330–5340.

[106] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. 2018. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=SkhQHMW0W.

[107] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sanchez. 2017. A survey on deep learning in medical image analysis. *Medical Image Analysis* 42 (2017), 60–88. DOI: https://doi.org/10.1016/j.media.2017.07.005

[108] Jiayi Liu, Jayanta Dutta, Nanxiang Li, Unmesh Kurup, and Mohak Shah. 2018. Usability study of distributed deep-learning frameworks for convolutional neural networks. In *Proceedings of the Deep Learning Day at SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'18)*.

[109] Gang Luo. 2016. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Netw. Model. Anal. Health Info. Bioinfo.* 5, 1 (23 May 2016), 18. DOI: https://doi.org/10.1007/s13721-016-0125-6

[110] Luo Mai, Chuntao Hong, and Paolo Costa. 2015. Optimizing network performance in distributed machine learning. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'15)*. USENIX Association, Santa Clara, CA. Retrieved from https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/mai.

[111] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'10)*. ACM, New York, NY, 135–146. DOI: https://doi.org/10.1145/1807167.1807184

[112] Dominic Masters and Carlo Luschi. 2018. Revisiting small batch training for deep neural networks. Retrieved from arxiv:1804.07612 http://arxiv.org/abs/1804.07612.

[113] Ruben Mayer, Christian Mayer, and Larissa Laich. 2017. The tensorflow partitioning and scheduling problem: It's the critical path!. In *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning (DIDL'17)*. ACM, New York, NY, 1–6. DOI: https://doi.org/10.1145/3154842.3154843

[114] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS'17)*. Retrieved from http://arxiv.org/abs/1602.05629.

[115] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2016. MLlib: Machine learning in apache spark. *J. Mach. Learn. Res.* 17, 34 (2016), 1–7. Retrieved from http://jmlr.org/papers/v17/15-237.html.

[116] H. Miao, A. Li, L. S. Davis, and A. Deshpande. 2017. Towards unified data and lifecycle management for deep learning. In *Proceedings of the IEEE 33rd International Conference on Data Engineering (ICDE'17)*. 571–582. DOI:https://doi.org/10.1109/ICDE.2017.112

[117] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. 2018. A hierarchical model for device placement. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=Hkc-TeZ0W.

[118] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 2430–2439. Retrieved from http://proceedings.mlr.press/v70/mirhoseini17a.html.

[119] Volodymyr Mnih and Geoffrey E. Hinton. 2012. Learning to label aerial images from noisy data. In *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*. 567–574.

[120] Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael I. Jordan. 2016. Sparknet: Training deep networks in spark. In *Proceedings of International Conference on Learning Representations. arXiv preprint arXiv:1511.06051.*

[121] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A distributed framework for emerging AI applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, Carlsbad, CA, 561–577. Retrieved from https://www.usenix.org/conference/osdi18/presentation/moritz.

[122] Apache MXNet. Distributed Training in MXNet. Retrieved from https://mxnet.incubator.apache.org/versions/master/faq/distributed_training.html.

[123] Apache MXNet. Does mxnet support Stale Synchronous Parallel (aka. SSP). Retrieved from https://github.com/apache/incubator-mxnet/issues/841.

[124] Apache MXNet. Gradient Compression : mxnet documentation. Retrieved from https://mxnet.incubator.apache.org/versions/master/faq/gradient_compression.html.

[125] Apache MXNet. MXNet. Retrieved from https://mxnet.apache.org/.

[126] Apache MXNet. MXNet Graph Optimization and Quantization based on subgraph and MKL-DNN. Retrieved from https://cwiki.apache.org/confluence/display/MXNET/MXNet+Graph+Optimization+and+Quantization+based+on+subgraph+and+MKL-DNN.

[127] Deepak Narayanan, Keshav Santhanam, Amar Phanishayee, and Matei Zaharia. 2018. Accelerating deep-learning workloads through efficient multi-model execution. In *Proceedings of the NIPS Workshop on Systems for Machine Learning*. Retrieved from https://www.microsoft.com/en-us/research/publication/accelerating-deep-learning-workloads-through-efficient-multi-model-execution/.

[128] Tencent. Tencent ncnn. Retrieved from https://github.com/Tencent/ncnn.

[129] Intel. Neon. Retrieved from https://github.com/NervanaSystems/neon.

[130] Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. 2018. A performance evaluation of federated learning algorithms. In *Proceedings of the 2nd Workshop on Distributed Infrastructures for Deep Learning (DIDL'18)*. ACM, New York, NY, 1–8. DOI:https://doi.org/10.1145/3286490.3286559

[131] Nils Nilsson. 2010. *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press. DOI:https://doi.org/10.1017/CBO9780511819346

[132] Robert Nishihara, Philipp Moritz, Stephanie Wang, Alexey Tumanov, William Paul, Johann Schleier-Smith, Richard Liaw, Mehrdad Niknami, Michael I. Jordan, and Ion Stoica. 2017. Real-time machine learning: The missing pieces. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS'17)*. ACM, New York, NY, 106–110. DOI:https://doi.org/10.1145/3102980.3102998

[133] Cyprien Noel and Simon Osindero. 2014. Dogwild!-Distributed hogwild for CPU & GPU. In *Proceedings of the NIPS Workshop on Distributed Machine Learning and Matrix Computations*.

[134] Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, Wei Wang, Qingchao Cai, Gang Chen, Jinyang Gao, Zhaojing Luo, Anthony K. H. Tung, Yuan Wang, Zhongle Xie, Meihui Zhang, and Kaiping Zheng. 2015. SINGA: A distributed

deep-learning platform. In *Proceedings of the 23rd ACM International Conference on Multimedia (MM'15)*. ACM, New York, NY, 685–688. DOI: https://doi.org/10.1145/2733373.2807410

[135] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric Chung. 2015. Accelerating Deep Convolutional Neural Networks Using Specialized Hardware. Retrieved from https://www.microsoft.com/en-us/research/publication/accelerating-deep-convolutional-neural-networks-using-specialized-hardware/.

[136] Or Ozeri, Effi Ofer, and Ronen Kat. 2018. Object storage for deep-learning frameworks. In *Proceedings of the 2nd Workshop on Distributed Infrastructures for Deep Learning (DIDL'18)*. ACM, New York, NY, 21–24. DOI: https://doi.org/10.1145/3286490.3286562

[137] PaddlePaddle. PaddlePaddle. Retrieved from http://paddlepaddle.org/.

[138] Jay H. Park, Sunghwan Kim, Jinwon Lee, Myeongjae Jeon, and Sam H. Noh. 2019. Accelerated training for CNN distributed deep learning through automatic resource-aware layer placement. *arXiv preprint arXiv:1901.05803* (2019).

[139] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *Proceedings of the Conference and Workshop on Neural Information Processing Systems (NIPS-W'17)*.

[140] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* 12 (Oct. 2011), 2825–2830.

[141] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An efficient dynamic resource scheduler for deep-learning clusters. In *Proceedings of the 13th EuroSys Conference (EuroSys'18)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3190508.3190517

[142] Christian Pinto, Yiannis Gkoufas, Andrea Reale, Seetharami Seelam, and Steven Eliuk. 2018. Hoard: A distributed data caching system to accelerate deep-learning training on the cloud. Retrieved from http://arxiv.org/abs/1812.00669.

[143] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2018. A survey on deep-learning: Algorithms, techniques, and applications. *ACM Comput. Surv.* 51, 5 (Sept. 2018). DOI: https://doi.org/10.1145/3234150

[144] OpenMined. OpenMined/PySyft: A library for encrypted, privacy preserving deep learning. Retrieved from https://github.com/OpenMined/PySyft.

[145] PyTorch Community. Multiprocessing Best Practies—PyTorch Master Documentation. Retrieved from https://pytorch.org/docs/stable/notes/multiprocessing.html.

[146] Aurick Qiao, Abutalib Aghayev, Weiren Yu, Haoyang Chen, Qirong Ho, Garth A. Gibson, and Eric P. Xing. 2018. Litz: Elastic framework for high-performance distributed machine learning. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 631–644. https://www.usenix.org/conference/atc18/presentation/qiao.

[147] Facebook. QNNPACK: Open-source library for optimized mobile deep learning. Retrieved from https://code.fb.com/ml-applications/qnnpack/.

[148] Jeff Rasley, Yuxiong He, Feng Yan, Olatunji Ruwase, and Rodrigo Fonseca. 2017. HyperDrive: Exploring hyperparameters with POP scheduling. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference (Middleware'17)*. ACM, New York, NY, 1–13. DOI: https://doi.org/10.1145/3135974.3135994

[149] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*. MIT Press, 693–701.

[150] Probir Roy, Shuaiwen Leon Song, Sriram Krishnamoorthy, Abhinav Vishnu, Dipanjan Sengupta, and Xu Liu. 2018. NUMA-caffe: NUMA-aware deep-learning neural networks. *ACM Trans. Archit. Code Optim.* 15, 2 (June 2018). DOI: https://doi.org/10.1145/3199605

[151] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. Retrieved from http://arxiv.org/abs/1609.04747.

[152] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533.

[153] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. 2018. A generic framework for privacy preserving deep learning. Retrieved from http://arxiv.org/abs/1811.04017.

[154] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R. Aberger, Kunle Olukotun, and Christopher Ré. 2018. High-accuracy low-precision training. Retrieved from http://arxiv.org/abs/1803.03383.

[155] SAS. SAS. Retrieved from https://www.sas.com/.

[156] Jurgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Netw.* 61 (2015), 85–117. DOI: https://doi.org/10.1016/j.neunet.2014.09.003

[157] John R. Searle. 1980. Minds, brains, and programs. *Behav. Brain Sci.* 3, 3 (1980), 417–424.

[158] Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, New York, NY, 2135–2135. DOI: https://doi.org/10.1145/2939672.2945397

[159] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs. In *Proceedings of Interspeech*. Retrieved from https://www.microsoft.com/en-us/research/publication/1-bit-stochastic-gradient-descent-and-application-to-data-parallel-distributed-training-of-speech-dnns/.

[160] Alexander Sergeev and Mike Del Balso. 2018. Horovod: Fast and easy distributed deep learning in TensorFlow. Retrieved from http://arxiv.org/abs/1802.05799.

[161] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. 2018. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems*. MIT Press, 10435–10444.

[162] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. ACM, New York, NY, 1310–1321. DOI: https://doi.org/10.1145/2810103.2813687

[163] Alex Smola. [n.d.]. What is the Parameter Server? Retrieved from https://www.quora.com/What-is-the-Parameter-Server.

[164] Alexander Smola and Shravan Narayanamurthy. 2010. An architecture for parallel topic models. *Proc. VLDB Endow.* 3, 1–2 (Sept. 2010), 703–710. DOI: https://doi.org/10.14778/1920841.1920931

[165] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 926–934. Retrieved from http://papers.nips.cc/paper/5028-reasoning-with-neural-tensor-networks-for-knowledge-base-completion.pdf.

[166] Evan R. Sparks, Ameet Talwalkar, Daniel Haas, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. 2015. Automating model search for large scale machine learning. In *Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC'15)*. ACM, New York, NY, 368–380. DOI: https://doi.org/10.1145/2806777.2806945

[167] Nikko Strom. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In *Proceedings of the 16th Annual Conference of the International Speech Communication Association*.

[168] Sainbayar Sukhbaatar and Rob Fergus. 2014. Learning from noisy labels with deep neural networks. Retrieved from http://arxiv.org/abs/1406.2080.

[169] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*.

[170] Zeyi Tao and Qun Li. 2018. eSGD: Communication efficient distributed deep learning on the edge. In *Proceedings of the USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*. USENIX Association, Boston, MA. Retrieved from https://www.usenix.org/conference/hotedge18/presentation/tao.

[171] TensorFlow. Distributed Training in TensorFlow. Retrieved from https://www.tensorflow.org/guide/distribute_strategy. https://www.tensorflow.org/guide/distribute_strategy.

[172] TensorFlow. Post-training quantization. Retrieved from https://www.tensorflow.org/lite/performance/post_training_quantization.

[173] TensorFlow. TensorFlow Federated. Retrieved from https://www.tensorflow.org/federated.

[174] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: A next-generation open-source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The 29th Annual Conference on Neural Information Processing Systems (NIPS'15)*. Retrieved from http://learningsys.org/papers/LearningSys_2015_paper_33.pdf.

[175] Leslie G. Valiant. 1990. A bridging model for parallel computation. *Commun. ACM* 33, 8 (Aug. 1990), 103–111. DOI: https://doi.org/10.1145/79173.79181

[176] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. 2011. Improving the speed of neural networks on CPUs. In *Proceedings of the Deep Learning and Unsupervised Feature Learning Workshop (NIPS'11)*.

[177] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. ModelDB: A system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA'16)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2939502.2939516

[178] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC'13)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2523616.2523633

[179] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys'15)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2741948.2741964

[180] Abhinav Vishnu, Charles Siegel, and Jeffrey Daily. 2016. Distributed TensorFlow with MPI. Retrieved from http://arxiv.org/abs/1603.02339.

[181] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou. 2017. DLAU: A scalable deep-learning accelerator unit on FPGA. *IEEE Trans. Comput.-Aid. Design Integr. Circ. Syst.* 36, 3 (Mar. 2017), 513–517. DOI: https://doi.org/10.1109/TCAD.2016.2587683

[182] Minjie Wang, Tianjun Xiao, Jianpeng Li, Jiaxing Zhang, Chuntao Hong, and Zheng Zhang. 2014. Minerva: A scalable and highly efficient training platform for deep learning. In *Proceedings of the NIPS Workshop on Distributed Machine Learning and Matrix Computations*.

[183] Minjie Wang, Hucheng Zhou, Minyi Guo, and Zheng Zhang. 2014. A scalable and topology configurable protocol for distributed parameter synchronization. In *Proceedings of 5th Asia-Pacific Workshop on Systems (APSys'14)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2637166.2637231

[184] Shaoqi Wang, Wei Chen, Aidi Pi, and Xiaobo Zhou. 2018. Aggressive synchronization with partial processing for iterative ml jobs on clusters. In *Proceedings of the 19th International Middleware Conference (Middleware'18)*. ACM, New York, NY, 253–265. DOI: https://doi.org/10.1145/3274808.3274828

[185] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database meets deep learning: Challenges and opportunities. *SIGMOD Rec.* 45, 2 (Sept. 2016), 17–22. DOI: https://doi.org/10.1145/3003665.3003669

[186] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter Pietzuch. 2016. Ako: Decentralised deep learning with partial gradient exchange. In *Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC'16)*. ACM, New York, NY, 84–97. DOI: https://doi.org/10.1145/2987550.2987586

[187] Jinliang Wei, Wei Dai, Aurick Qiao, Qirong Ho, Henggang Cui, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. 2015. Managed communication and consistency for fast data-parallel iterative analytics. In *Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC'15)*. ACM, New York, NY, 381–394. DOI: https://doi.org/10.1145/2806777.2806778

[188] Markus Weimer, Yingda Chen, Byung-Gon Chun, Tyson Condie, Carlo Curino, Chris Douglas, Yunseong Lee, Tony Majestro, Dahlia Malkhi, Sergiy Matusevych, Brandon Myers, Shravan Narayanamurthy, Raghu Ramakrishnan, Sriram Rao, Russel Sears, Beysim Sezgin, and Julia Wang. 2015. REEF: Retainable evaluator execution framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*. ACM, New York, NY, 1343–1355. DOI: https://doi.org/10.1145/2723372.2742793

[189] wekadl4j [n.d.]. Retrieved from https://github.com/Waikato/wekaDeeplearning4j.

[190] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*. MIT Press, 1509–1519.

[191] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. Retrieved from http://arxiv.org/abs/1609.08144.

[192] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2019. A comprehensive survey on graph neural networks. Retrieved from http://arxiv.org/abs/1901.00596.

[193] xgboost [n.d.]. XGBoost. Retrieved from https://xgboost.ai/.

[194] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. 2015. Learning from massive noisy labeled data for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*.

[195] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, Carlsbad, CA, 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao

[196] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Trans. Big Data* 1, 2 (June 2015), 49–67. DOI: https://doi.org/10.1109/TBDATA.2015.2472014

[197] Feng Yan, Olatunji Ruwase, Yuxiong He, and Trishul Chilimbi. 2015. Performance modeling and scalability optimization of distributed deep-learning systems. In *Proceedings of the 21th ACM SIGKDD International Conference*

*on Knowledge Discovery and Data Mining (KDD'15)*. ACM, New York, NY, 1355–1364. DOI: https://doi.org/10.1145/2783258.2783270

[198] T. Young, D. Hazarika, S. Poria, and E. Cambria. 2018. Recent trends in deep-learning-based natural language processing [review article]. *IEEE Comput. Intell. Mag.* 13, 3 (Aug. 2018), 55–75. DOI: https://doi.org/10.1109/MCI.2018.2840738

[199] Yuan Yu, Martín Abadi, Paul Barham, Eugene Brevdo, Mike Burrows, Andy Davis, Jeff Dean, Sanjay Ghemawat, Tim Harley, Peter Hawkins, Michael Isard, Manjunath Kudlur, Rajat Monga, Derek Murray, and Xiaoqiang Zheng. 2018. Dynamic control flow in large-scale machine learning. In *Proceedings of the 13th EuroSys Conference (EuroSys'18)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3190508.3190551

[200] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*. USENIX, San Jose, CA, 15–28. Retrieved from https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.

[201] C. Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2016. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'16)*. 1–8. DOI: https://doi.org/10.1145/2966986.2967011

[202] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'15)*. ACM, New York, NY, 161–170. DOI: https://doi.org/10.1145/2684746.2689060

[203] C. Zhang, H. Tian, W. Wang, and F. Yan. 2018. Stay fresh: Speculative synchronization for fast distributed machine learning. In *Proceedings of the IEEE 38th International Conference on Distributed Computing Systems (ICDCS'18)*. 99–109. DOI: https://doi.org/10.1109/ICDCS.2018.00020

[204] H. Zhang, C. Hsieh, and V. Akella. 2016. HogWild++: A new mechanism for decentralized asynchronous stochastic gradient descent. In *Proceedings of the IEEE 16th International Conference on Data Mining (ICDM'16)*. 629–638. DOI: https://doi.org/10.1109/ICDM.2016.0074

[205] Haoyu Zhang, Logan Stafman, Andrew Or, and Michael J. Freedman. 2017. SLAQ: Quality-driven scheduling for distributed machine learning. In *Proceedings of the Symposium on Cloud Computing (SoCC'17)*. ACM, New York, NY, 390–404. DOI: https://doi.org/10.1145/3127479.3127490

[206] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P. Xing. 2017. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *Proceedings of the USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'17)*. USENIX Association, Berkeley, CA, 181–193. Retrieved from http://dl.acm.org/citation.cfm?id=3154690.3154708.

[207] Huasha Zhao and John Canny. [n.d.]. Butterfly mixing: Accelerating incremental-update algorithms on clusters. In *Proceedings of the SIAM International Conference on Data Mining*. 785–793. DOI: https://doi.org/10.1137/1.9781611972832.87 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972832.87

[208] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. 2016. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*.

[209] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. Retrieved from http://arxiv.org/abs/1606.06160.

[210] Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*.

[211] Yongqiang Zou, Xing Jin, Yi Li, Zhimao Guo, Eryu Wang, and Bin Xiao. 2014. Mariana: Tencent deep-learning platform and its applications. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1772–1777. DOI: https://doi.org/10.14778/2733004.2733082

Received March 2019; revised July 2019; accepted September 2019