Dehghani and Yazdanparast
Journal of Big Data (2023) 10:158
https://doi.org/10.1186/s40537-023-00829-x

Journal of Big Data

RESEARCH Open Access

From distributed machine to distributed deep learning: a comprehensive survey

Mohammad Dehghani[1*] and Zahra Yazdanparast[2]

*Correspondence:
dehghani.mohammad@ut.ac.ir

[1] University of Tehran, Tehran, Iran
[2] Tarbiat Modares University, Tehran, Iran

Abstract

Artificial intelligence has made remarkable progress in handling complex tasks, thanks to advances in hardware acceleration and machine learning algorithms. However, to acquire more accurate outcomes and solve more complex issues, algorithms should be trained with more data. Processing this huge amount of data could be time-consuming and require a great deal of computation. To address these issues, distributed machine learning has been proposed, which involves distributing the data and algorithm across several machines. There has been considerable effort put into developing distributed machine learning algorithms, and different methods have been proposed so far. We divide these algorithms in classification and clustering (traditional machine learning), deep learning and deep reinforcement learning groups. Distributed deep learning has gained more attention in recent years and most of the studies have focused on this approach. Therefore, we mostly concentrate on this category. Based on the investigation of the mentioned algorithms, we highlighted the limitations that should be addressed in future research.

Keywords: Artificial intelligence, Machine learning, Distributed machine learning, Distributed deep learning, Ditributed reinforcement learning, Data-parallelism, Model-parallelism

Introduction

Artificial intelligence (AI) is a rapidly developing field that uses knowledge to simulate human behaviors [1] and train computers to learn, make judgments, and make decisions similarly to humans [2, 3]. In other words, AI involves developing techniques and algorithms that are capable of thinking, acting, and implementing tasks using protocols that are otherwise beyond human comprehension [4].

Machine learning (ML) is a subset of AI that learns from historical data, without being explicitly programmed [5]. ML algorithms can be used to analyze data and build data-driven systems, including classification, clustering, regression, association rule mining, and reinforcement learning [6, 7]. Deep learning is a branch of machine learning that uses artificial neural networks to intelligently analyze large amounts of data [8, 9]. The digital world has been provided with a large amount of data in many different areas, such as cybersecurity, business, health, and the Internet of Things (IoT). This wealth of data is analyzed by machine learning, one of the most popular

technologies in Industry 4.0, in order to develop smarter and more automated systems [10]. ML algorithms have been widely used in many application domains, such as computer vision [11, 12] and natural language processing (NLP) [13]. They have proven to be effective across a wide range of industries comprising education [14], healthcare [15], marketing [16], transportation [17], energy [18], combustion science [19], and manufacturing [20].

A machine learning solution is influenced by the characteristics of the data and the performance of the learning algorithms [10]. Traditionally, a bottleneck in developing more intelligent systems was data availability which is no longer the case. However, the problem now is that learning algorithms are unable to employ all the data within a reasonable period of time for learning. Creating an effective ML model is generally a complex and time-consuming process that involves selecting an appropriate algorithm and developing an optimal model architecture [21].

In order to train ML models over large volumes of data, one machine's storage and computation capabilities are insufficient. One solution for this challenge is employing distributed machine learning for the executation of ML programs on clusters, data centers, and cloud providers [22]. It can divide the learning process across several workstations, achieving scalability of learning algorithms [23]. Generally, distributed machine learning allows a cloud or server to collect combined models from multiple participants, with each participant training their own model locally [24]. Furthermore, it allows the handling of naturally distributed data sets, which is a common scenario in many real-world applications [23]. Many researchers are currently focusing on algorithmic correctness and faster convergence rates of ML models [25, 26].

This paper presents a comprehensive view of various types of distributed machine learning algorithms to overcome the problems of traditional machine learning approaches. The main contribution of this study is the introduction of different distributed machine learning techniques, which can provide a valuable guide for those in academia and industry interested in studying, researching, and developing machine learning-based data-driven automation and intelligent systems.

Compared to traditional machine learning algorithms, distributed deep learning has received more attention in studies. In recent years, deep learning has achieved tremendous success in various areas, such as image processing, NLP, and speech recognition. One of the reasons for this success is the availability of a large amount of data and the increased size of the deep learning model. As deep learning continues to improve, increasing its scalability is essential. Currently, distributed deep learning is gaining increasing recognition to overcome the challenges of deep learning. The benefits of distributed deep learning include [27]:

- Increased scalability: Distributed deep learning is increasingly necessary as neural networks and datasets grow. The training process could be scaled to handle more extensive datasets and models, and more computers can be added to the distributed system to increase scalability and enable quicker training cycles.
- Resource utilization: By spreading the work across several computers, we may use the already available resources and reach a higher level of parallelism, resulting in shorter training durations and less resource usage.

The rest of the paper is organized as follows. Section "Related works" provides an overview of differences between this survey and existing surveys. Section "Distributed machine learning algorithms" provides a taxonomy of distributed machine learning algorithms. Section "Conclusions and research directions" concludes the survey and considers the direction of future research.

Related works

To the best of the authors' knowledge, very few works have proposed to survey distributed algorithms. This article presents a literature review on distributed machine learning algorithms and states their distinctions from existing surveys. This study provides an extensive overview of the current state-of-the-art in the field by outlining the challenges and opportunities of distributed machine learning over conventional (centralized) machine learning and discussing the techniques used for distributed machine learning.

Verbraeken et al. [28] discussed distributed techniques and systems and covered various aspects of distributed machine learning including its challenges and opportunities. Langer et al. [29] investigated the fundamental principles for training deep neural network (DNN) in a cluster of independent machines. Moreover, they analyzed the common attributes of training deep learning models and surveyed the distribution of such workloads in a cluster to attain collaborative model training. Ouyang et al. [30] provided a comprehensive survey of communication strategies for distributed DNN. They considered algorithm and network optimizations to diminish the communication overhead in distributed DNN training. Tang et al. [31] have detailed discussions on communication compression techniques.

Xing et al. [22] investigated different synchronization schemes, scheduling, workload balancing schemes, and communication types. Nassef et al. [32] overviewed distributed machine learning architectures for 5G networks and beyond. They also considered optimizing communication, computation, and resource distribution to improve the performance of distributed machine learning in 5G networks. Mayer et al. [33] overviewed deep learning infrastructures, parallel deep learning methods, scheduling, and data management. Muscinelli et al. [34] surveyed the influential research of distributed learning technologies playing a critical role in the 6G networks. In particular, they reviewed federated learning and multi-agent reinforcement learning algorithms. They discussed several emerging applications and their fundamental concepts relating to their implementation and optimization.

An overview of distributed deep reinforcement learning is provided by Yin et al. [35]. Their study compared classical distributed deep reinforcement learning (DRL) methods and examined important factors that contribute to efficient distributed learning. Furthermore, they discuss both single player, single agent distributed DRLs as well as multiple players, multiple agents. A further review was carried out of recently released toolboxes which assist in the realization of distributed DRL. Antwi-Boasiako et al. [36] provided a comprehensive survey of privacy issues in distributed deep learning. They described various cryptographic algorithms and other techniques that can be used to preserve privacy, as well as their benefits and drawbacks.

Other surveys lack detailed discussions on the distribution of machine learning algorithms, which are introduced explicitly in this survey. We cover a variety of algorithms

of traditional machine learning (classification and clustering algorithms), deep learning, and reinforcement learning.

Distributed machine learning algorithms

A number of popular algorithms for distributed machine learning are described in this section. Figure 1 represents a classification of these algorithms.

Distributed classification

Classification is a supervised method in machine learning that uses label data to predict the future. A classification model trains using historical data to produce a function that can predict outputs for new unseen data. Classification outputs are categorical [37]. Two popular distributed classification methods are boosting and support vector machines (SVM), which we discuss in the following. These algorithms are summarized in Table 1.

Distributed boosting

Boosting is a technique used in machine learning that trains an ensemble of so-called weak learners to produce an accurate model, or strong learner. It relies on the idea that learning and combining many weak classifiers instead of learning a single strong classifier can achieve better performance [38].

In distributed boosting, AdaBoost [39] is modified for use in distributed environments [23]. Lazarevic and Obradovic [40] proposed a framework for the integration of specialized classifiers learned from very large, distributed datasets that cannot be stored in the main memory of a computer. The suggested method combines classifiers from all sites and creates a classifier ensemble on each site. Filmus et al. [41] proposed a distributed boosting algorithm that is resilient to a limited amount of noise using Impagliazzo's hard-core lemma [42]. This algorithm is novel because it applies non-standard boosting that identifies small "hard" sets so that any hypothesis derived from the class will have high error rates.

Sarnovsky and Vronc [43] implemented a distributed boosting algorithm based on MapReduce and employed the GridGain framework for distributed data processing. The algorithm works as follows: (i) The master node computes the dataset statistics,

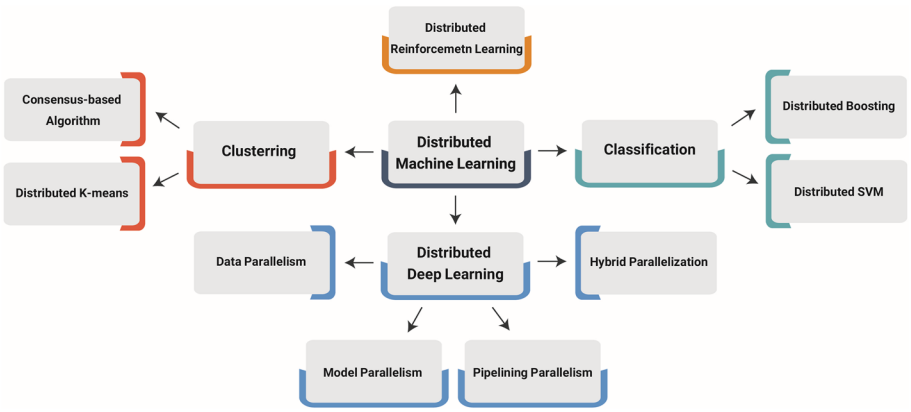

Fig. 1 Ditributed machin learning algorithms

Table 1 Distributed classification

Algorithm	Articles	Year	No. of references	Simulation/ Dataset	Evaluation metrics
Distributed Boosting	[41]	2022	26		• Accuracy • Correctness • Communication complexity
	[44]	2017	20	• ocr17 • ocr49 • forestcover12 • particle • ringnorm • twonorm • Yahoo!	• Error
	[43]	2014	19	• Reuters-21,578 • Medlin	• Time
	[40]	2002	31	• Covertype • Pen-based digits • Waveform • LED	• Accuracy • Speedup
Distributed SVM	[51]	2019	32	• Optical satellite images	• RMSE
	[50]	2015	14	• Spiral data set • MNIST • COVERTYPE	• Integrations • Parallel Speed-up
	[49]	2011	40	• Images from Corel database	• accuracy • training times
	[48]	2008	17	• MNIST	• CPU seconds • Number of iterations • Communication overhead
	[47]	2003	16	• Handwritten Chinese database ETL9B	• Error rate

including the number of categories and the distribution of indexed terms within the dataset. (ii) The number of subtasks (jobs) necessary to construct the final model is calculated based on the gridSize parameter. (iii) AdaBoost is implemented to train the model.

Cooper et al. [44] presented two distributed boosting algorithms. The first algorithm uses the entire dataset to train a classifier and requires significant communication between the distributed sites. In contrast, the second algorithm requires little communication but trains its final classifier using a subset of the dataset.

Distributed SVM

SVM is a linear binary classifier that identifies a single boundary between classes. It seeks to overcome the local extremum dilemma inherent in other machine learning techniques by solving a convex quadratic optimization problem [45]. SVM determines an optimal hyperplane (a line in the simplest case) by which it is possible to divide the dataset into a discrete number of classes. To optimize the separation, SVM employs a portion of the training sample that lies closest to the optimal decision boundary in the feature space [46].

SVM training requires quadratic computation time. To its speed up, several distributed computing paradigms have been investigated by dividing the training dataset into smaller sections and processing each section in the parallel cluster of computers [47]. Lu et al. [48] proposed a distributed parallel support vector machine (DPSVM)

that exchanges support vectors among a network of strongly connected servers. It results in limited communication costs and fast training times for multiple servers working concurrently on distributed datasets. DPSVM uses SVM as a local classification mechanism for subsets of training data within a strongly connected network.

According to Alham et al. [49], SVM training is computationally intensive, particularly with large datasets. To address this issue, they presented MRSMO, a distributed SVM algorithm based on MapReduce for automatic image annotation. MRSM partitions datasets into small subsets and optimizes them across a cluster of computers. Ke et al. [50] proposed a method for distributed SVM, where the local SVMs use the state-of-the-art SVM solvers and implement it on MapReduce to shorten the communication between nodes. Wang et al. [51] described a spatially distributed SVM method for estimating shallow water bathymetry from optical satellite imagery. This method uses SVMs that have been trained locally and spatially weighted votes to make predictions. According to the results, the localized model reduced the RMSE by 60%.

Distributed clustering

Clustering is an unsupervised machine learning method that involves defining classes from data without knowing the labels of classes. In clustering, data is categorized into collections (or clusters) based on their similarities [52]. Clustering algorithms apply when there is no class for prediction, so the instances divide into natural groups. Clustering distributed classifiers relies on the following: (i) A measure of classifier distance, (ii) An efficient algorithm for computing this distance measure for classifiers induced in physically distributed databases, and (iii) A clustering algorithm [23]. The distributed clustering algorithms of consensus-based algorithm and distributed k-means algorithm are discussed in the following. A summary of these algorithms can be found in Table 2.

Table 2 Distributed clustering

Algorithm	Articles	Year	No. of references	Simulation/ Dataset	Evaluation metrics
Consensus-based algorithm	[53]	2016	35	• Wireless sensor networks (WSNs)	• Within-cluster sum of squares (WCSS) • Iteration time
	[54]	2011	15	• Two data sites	• Xie-Beni (XB) fuzzy clustering validity index
Distributed k-means	[59]	2021	25	• MRI image segmentation	• Number of iterations
	[61]	2016	39	• YearPredictionMSD	• Communication costs
	[57]	2013	33	• Mammal's Milk • River dataset • Water treatment dataset	• Communication Overhead • Computation Overhead
	[58]	2013	42	• Wireless sensor networks	• Time complexity • Memory complexity
	[60]	2008	38	• P2P network	• Accuracy • Scalability • Communication

Consensus-based algorithm

Consensus clustering is a technique in which multiple clusters combine into a more stable single cluster that is better than the input clusters. It yields a stable and robust final clustering in agreement with multiple clusterings. Consensus clustering is a more robust approach that relies on multiple iterations of the chosen clustering method on sub-samples of the dataset [53]. Vendramin et al. [54] presented a consensus-based algorithm for distributed fuzzy clustering that allows an automatic estimation of the number of clusters by using a distributed version of the Xie-Beni validity criterion.

Distributed k-means

K-means clustering is one of the most popular clustering algorithms due to its many advantages, such as simple mathematical concepts, quick convergence, and ease of implementation [55]. K-means is an iterative process in which k centroids are determined. Then, each sample is assigned to the closest current centroid (assignment phases). The new centroid will be determined by the average of all samples in the same partition (refinement phase) [56].

Patel et al. [57] presented a parallel version of k-means focusing on privacy preservation. In distributed environments, where data mining becomes a collaborative effort, it is crucial to maintain privacy. The basic concept involves the use of a secret sharing mechanism to share information privately along with a code-based zero-knowledge identification scheme to add protection against malicious adversaries. Oliva et al. [58] suggested a fully distributed execution of the k-means clustering algorithm. It was applied for wireless sensor networks where each agent was provided with a high-dimensional observation. To spread information on current centroids across the network, the proposed algorithm uses a maximum consensus algorithm. Each agent employs this information to select the nearest centroid, thus segmenting the network into communities. For the purpose of updating centroids, meta-information is gathered by combining max-consensus and average-consensus algorithms. The agents are able to update the centroids locally once such information has been gathered.

A distributed k-means method has proposed by Benchara and Youssfi [59]. It integrates a parallel virtual distributed computing model with a low-cost communication mechanism. K-means is implemented as a distributed service using an asynchronous communication protocol based on Advanced Message Queuing Protocol (AMQP). Datta et al. [60] discussed a distributed k-means clustering, in which data and computing resources are distributed over a large peer-to-peer network. Using two algorithms, it approximates the result produced by the centralized k-means clustering algorithm. The first algorithm is intended to be used in a dynamic peer-to-peer network. It is capable of producing clusterings only through the use of "local" synchronization. In the second algorithm, peers are uniformly sampled, and analytical guarantees are provided about the accuracy of clustering on an Internet-based peer-to-peer system. Ding et al. [61] studied distributed k-means clustering, in which dimensions of the data are distributed across multiple computers.

Distributed deep learning

Deep learning is a type of machine learning process that uses interconnected nodes or neurons in a layered structure that resembles the human brain [62]. Neural networks consist of many computation units, known as neurons, which are connected and form the neural network. The input neurons of the network are actuated by input parameters. The neurons in the following layer are activated by weighted connections from neurons in the previous layer. To provide the desired functionality, usually classification or regression, the neural network must determine the appropriate weight value for every connection [63].

To overcome the problem of training DNN models, which requires a large volume of data and computational resources, a variety of parallel and distributed methods have been proposed [64, 65]. These methods can be divided into four categories: data parallelism, model parallelism, pipeline parallelism, and hybrid parallelism [66]. An overview of these algorithms is presented in Table 3.

Data parallelism

Data parallelism is a popular method for training a neural network that involves sharing a large-scale DNN among all computational workers [67, 68]. In data parallelism, data samples are partitioned into mini-batches (Fig. 2) [69]. During the computation of gradients, each node or worker contains one of the mini-batches, a replica of the neural network model, and independently computes gradients (usually using the Mini-Batch SGD) [70]. The following steps are involved in training: (i) Computation of local gradients by each worker; (ii) Calculation of the new parameters of the DNN by combining all subgradients. (iii) Distribution of the new parameters among the workers, and retraining of the DNN [71]. To aggregate and update gradients, either a centralized architecture such as parameter server architecture [72], or a decentralized architecture such as All-Reduce [73] is used.

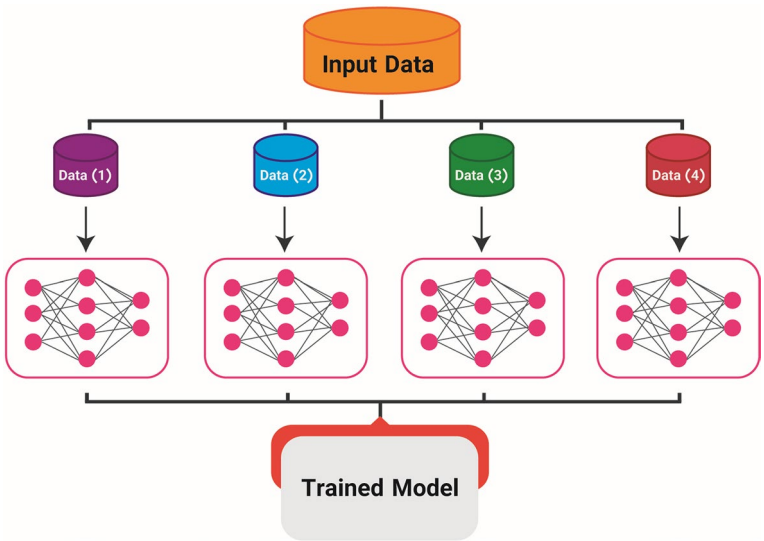

Fig. 2 Data parallelism

Data parallelization allows for processing large datasets that cannot be stored on a single machine and can increase the system's throughput through distributed parallel computing. However, data parallelism also has some challenges, including the overhead of parameter synchronization, optimization algorithms, and hardware limitations when the DNN model size is too large [64, 74].

Dean et al. [67] introduced DistBelief, a framework for parallel distributed training of deep networks, and two new algorithms for large-scale distributed training: Downpour SGD and Sandblaster L-BFGS. Downpour SGD is an asynchronous variant of stochastic gradient descent (SGD), which is effective for training nonconvex deep learning models. As a distributed version of L-BFGS, Sandblaster L-BFGS makes efficient use of network bandwidth to train a single model on a large number of concurrent cores.

Zhang et al. [68] developed an algorithm for optimizing deep learning under communication constraints in a parallel environment. In this algorithm, elastic force is used to link the parameters calculated by local workers to the central variable stored by the parameter server. As a result, the amount of communication between the master and the local workers is reduced. Asynchronous and synchronous variants of this algorithm are available.

An algorithm for distributed SGD based on a communication trigger mechanism has been proposed by George et al. [69]. They presented the Distributed Event-Triggered Stochastic GRAdient Descent (DETSGRAD) algorithm, which allows networked agents to update model parameters periodically in order to solve non-convex optimizations. The evaluation was conducted using MNIST with 60,000 images for training and 10,000 images for testing. During training, each agent used LeNet-5. There are two types of DETSGRAD: DETSGRAD-r, in which agents are randomly selected from the entire training set, and DETSGRAD-s, in which each agent has access to the images of only one class. According to the results obtained after 40 epochs with 10 agents, the accuracy of DETSGRAD-r and DETSGRAD-s was 98.33 and 98.51, almost similar to the accuracy of SGD-r and SGD-s with 98.97 and 98.87. Based on the results, it appears that DETSGRAD reduced inter-agent communication while maintaining similar performance.

Kim et al. [70] proposed Parallax, a framework that integrates parameter server and AllReduce architectures in order to optimize parallel data training by exploiting model parameter sparsity. ResNet-50 and Inception-v3 were used to classify images from the ImageNet dataset. In NLP, the LM model was trained using the One Billion Word Benchmark, and the NMT model was trained using the WMT English-German dataset. Image classification models are trained at the same speed as Horovod and 1.53x faster than TensorFlow. For NLP models, Parallax has achieved speedups of 2.8x and 6.02x compared to TensorFlow and Horovod.

The Dynamic Batch Size (DBS) strategy has been proposed by Ye et al. [71] for the distributed training of DNNs. According to the performance of previous epochs, DBS evaluates the performance of each worker, and then dynamically adjusts the batch size and dataset partition. DBS aims to optimize cluster utilization based on worker performance and can be used with all synchronous methods. The estimated batch size and dataset partition are employed in the next training. As compared synchronous

stochastic gradient descent (S-SGD), DBS saved approximately 12% of the consumed time of each epoch on a scale of 4 and 10% on a scale of 8. The decreases can be attributed to cluster synchronization and communication costs, which are higher as the cluster expands.

Using data parallelism, Dong et al. [75] proposed a technique called "natural compression" that is an effective method for compressing data. It is based on the randomized rounding to the nearest (negative or positive) power of two, which can be computed in a "natural" manner without taking into account the mantissa. The natural compression method reduced the training time for ResNet110 by 26% (compared to only a 9% decrease for QSGD for the same setup) and 66% for AlexNet, compared to using no compression. In their study, they also presented convergence theory for distributed SGD to apply bidirectional compression at both the master and worker levels.

Model parallelism

Model parallelism is a technique used to speed up the training of DNNs by dividing a large model among multiple nodes or workers (Fig. 3) [76]. Each node is responsible for part of the computation of model parameters, such as weights [74]. However, the major challenges are how to break the model into partitions, as each model has its own characteristics, and the allocation of partitions to GPUs to maximize the efficiency of training [77]. Furthermore, model parallelism alone is not scalable [78] due to high communication latency between devices.

A fully decoupled training scheme was proposed by Zhang et al. [79]. A neural network was broken down into several modules (K) and trained on multiple devices. In the WRN-28-10 (CIFAR-10) case, delayed gradients slightly outperformed the decoupled greedy learning and achieved a speedup of 1.88x for $K=2$, 2.72x for $K=3$, and 3.20x for $K=4$. For the ResNet-101 (ImageNet) case, the delayed gradients achieved a 1.68x speedup for $K=2$, and 2.1x and 2.3x for $K=3$ and $K=4$.

Huo et al. [80] proposed a Decoupled Parallel Back-propagation (DDG) algorithm for training feedforward neural networks. This algorithm splits the model and stores the delayed error gradient to solve the backward-locking problem. By increasing the number of GPUs from two to four, the method is able to reduce the total computation time by about 30–50%.

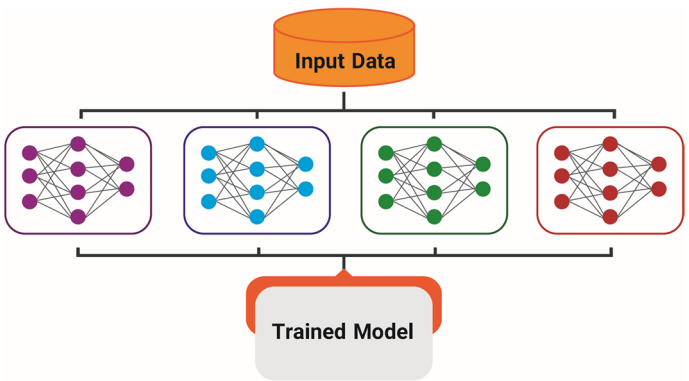

Fig. 3 Model parallelism

Table 3 Distributed deep learning

Algorithm	Articles	Year	No. of references	Simulation/ dataset	Evaluation metrics
Data parallelism	[75]	2022	61	• ResNet110 and AlexNet models on CIFAR10	• Train loss • Test accuracy
	[72]	2022	24	• Matrix Classification • MovieLens Avazu-CTR	• Convergence time per epoch • Disk I/O • Network communication
	[65]	2021	138	• ResNet-50 on ImageNet dataset • ALBERT-large on Wiki-Text-103 dataset	• Training time
	[71]	2020	37	• ResNet101 on CIFAR10 dataset	• Convergence • Robustness
	[69]	2019	53	• LeNet-5 on MNIST dataset	• Accuracy
	[70]	2019	46	• ResNet-50 and Inception-v3 on ImageNet • LM model on One Billion Word Benchmark • NMT model on WMT English-German dataset	• Validation error • Test perplexity • BLEU
	[73]	2018	20	• Inception V3 • ResNet-101 • VGG-16	• Images processed per second
	[68]	2015	31	• CNN on CIFAR and ImageNet datasets	• Test loss • Test error
	[67]	2012	29	• ImageNet	• Accuracy
Model parallelism	[77]	2021	72	• GNN model on OGB-Product, OGB-Paper, UK-2006-05, UK-Union, Facebook datasets	• ROC
	[79]	2021	29	• ResNet and WRN models on CIFAR-10 dataset • ResNet-18 and MobileNet v2 on Tiny-ImageNet	• Error rate
	[76]	2019	30	• AlexNet, Inception-v3 and ResNet-101 on ImageNet dataset • RNNTC on Movie Reviews dataset • RNNLM on Penn Treebank dataset • NMT on WMT English-German dataser	• Accuracy
	[80]	2018	25	• ResNet on CIFAR	• Accuracy

Table 3 (continued)

Algorithm	Articles	Year	No. of references	Simulation/ dataset	Evaluation metrics
Pipelining parallelism	[81]	2020	29	• AmoebaNet-D • U-Net	• Throughput • Speed up
	[82]	2019	57	• VGG-16 and ResNet-50 on ImageNet • AlexNet on Synthetic Data • GNMT-16 and GNMT-8 on WMT16 EN-De • AWD LM on Penn Treebank • S2VT on MSVD	• Accuracy • Speed up
	[83]	2018	50	• VGG16, ResNet-152, Inception v4 and SNN on CIFAR-10 • Transformer on IMDb Movie Review Sentiment Dataset • Residual LSTM on IMDb Dataset	• Speed up
	[84]	2017	25	• VGG-A model on ImageNet	• Speed up
Hybrid parallelization	[88]	2023	64	• MATCHNET, CTRDNN, 2EMB and NCE models	• Scheduling performance • Throughput
	[64]	2022	57	• 3D-ResAttNet on Alzheimer's Disease Neuroimaging Initiative (ADNI) database	• Speedup • Accuracy • Training time•
	[91]	2020	64	• CosmoFlow and 3D UNet models	• MSE
	[86]	2019	23	• Seq2Seq RNN MT with attention on WMT14 and WMT17 datasets	• BLEU scores
	[87]	2019	120	• SFC, SCONV, Lenet-c, Cifar-c, AlexNet,VGG-A, VGG-B, VGG-C, VGG-D and VGG-E models on MNIST, CIFAR-10 and ImageNet datasets	• Energy efficiency • Performance • Total communication
	[85]	2018	67	• AlexNet and VGG models	• Communication Overhead • Training time • Speed up
	[90]	2017	33	• CNN on ImageNet LSVRC-2010 dataset	• Error rate
	[89]	2013	12	• ImageNet dataset	• Error rate

Pipelining parallelism

Pipeline parallelism is a technique used to divide training tasks for DNN models into sequential processing stages, and the results of each sequence are passed on to the next [81]. Narayanan et al. [82] proposed PipeDream, a system that adds inter-batch pipelining to intra-batch parallelism to further improve parallel training throughput, allowing computation and communication to overlap more effectively and reduce communication. PipeDream updates model parameters for numerically correct gradient computations. In addition, forward and backward passes are scheduled concurrently on separate workers in order to minimize pipeline stalls. Furthermore, it automatically distributes DNN layers among workers so that work can be balanced and communication can be

minimized. Compared to common intra-batch parallelism techniques, PipeDream performed 5.3 times faster.

Chen et al. [83] achived robust training accuracy by implementing a pipelined model and using a novel weight prediction technique. On a four-GPU platform, this method achieves an 8.91x speedup compared with data parallelism. Lee et al. [84] implemented a thread in each computer server to overlap computation and communication problems during model training. They achieved speedups of 62.97x and 77.97x for training VGG-A model on ImageNet. In parallel pipelines, there are two major problems: the slowest stage becomes a bottleneck. and scalability is limited [64].

Hybrid parallelization

Hybrid parallelization is a technique employed to minimize the communication overhead of DNN training by combining data and model parallelization techniques [85]. Ono et al. [86] proposed a hybrid approach, which applies a model parallel to the RNN encoder-decoder in the Seq2Seq model, and data parallel to the attention-softmax. According to the results, using four GPUs increased training speed by 4.13 to 4.20 times over using one GPU alone. The solution proposed by Song et al. [87], HYPAR, partitions the feature map tensors (inputs and outputs), kernel tensors, gradient tensors, and error tensors among the DNN accelerators. During training, the goal of optimization is to search for a partition that minimizes the amount of communication. In an evaluation of classic Lenet to large-size model VGGs, HYPAR outperformed model parallelism and data parallelism alone. Compared to data parallelism, results showed a performance of 3.39x and an energy efficiency of 1.51x was achieved.

A hybrid parallelization method for training DNNs was proposed by Akintoye et al. [64], as well as a Genetic Algorithm Based Heuristic Resources Allocation (GABRA) approach for optimal partitioning on GPUs to maximize computing performance. Model parallelization includes neural network model partitioning and the GABRA mechanism. Asynchronous Stochastic Gradient Descent (ASGD) and ring All-Reduce mechanisms are used for data parallelization. The proposed approach that was applied to a 3D Residual Attention Deep Neural Network (3D-ResAttNet) using the ANDI dataset, achieved a 20% average improvement over existing parallel methods in terms of training time while maintaining accuracy.

The Heterogeneous Parameter Server (HeterPS) was proposed by Liu et al. [88] to facilitate the training of large-scale models using elastic heterogeneous computing resources. HeterPS consists of three modules: (i) Scheduling module for the DNN layer that generates a scheduling plan as well as a provisioning plan. In the provisioning plan, the number of computing resources of each type is defined, whereas in the scheduling plan, each layer is assigned the appropriate type of computing resource. (ii) A data management module that facilitates the transfer of data between clusters or servers. (iii) A distributed training module that exploits the combination of data parallelism and pipeline parallelism in order to parallelize the training process of the model. Experimental results indicated that the provisioning method can outperform baseline methods by up to 57.9% and the scheduling method can outperform state-of-the-art methods by up to 312.3% (monetary cost). Additionally, the framework has a throughput 14.5 times greater than TensorFlow.

Yadan et al. [89] achieved a speed improvement of 2.2x when training a large deep CNN using hybrid parallelization. Krizhevsky et al. [90] used hybrid parallelization to train a deep CNN and evaluated its performance by classifying 1.2 million images in the ImageNet LSVRC-2010 dataset. By combining parallel data and model training, Oyama et al. [91] increased throughput and minimized I/O scaling bottlenecks for a 3D CNN.

To address the challenge of aggregating sub-gradients effectively, several synchronous strategies have been used, including Parallel S-SGD [92, 93] and Bulk Synchronous Parallel (BSP) [94] among others.

Distributed deep reinforcement learning

Reinforcement learning is a learning algorithm that involves learning by interacting with the environment through actions, observations, and rewards. Reinforcement learning faces a major challenge when it comes to learning good representations of high-dimensional states or action spaces [95]. DRL combines reinforcement learning with deep learning, allowing the representation of a continuous state or action, which was difficult for a table representation [96]. However, DRL faces technical and scientific challenges such as data inefficiency, multi-task learning, and exploration-exploitation trade-offs. To overcome these challenges, distributed DRL was introduced. In distributed DRL, agents can run simultaneously on several computers allowing for parallelization of the learning process [97].

Nair et al. [98] introduced the GORILA (General Reinforcement Learning Architecture), which is similar to DQN [99], but with multiple workers and learners, and the SGD is computed using the DistBelief [67] method. In the GORILA architecture, there are N different actor processes, which are applied to N corresponding instances of the same environment. The Q-network is replicated in each actor, which determines its behavior. A parameter server periodically synchronizes the parameters of the Q-network. There are N learner processes in GORILA. Learners contain replicas of the Q-network and are responsible for computing desired changes to its parameters. There are many ways in which a reinforcement learning agent may be parallelized using the GORILA architecture. One approach is parallel acting, where large quantities of data can be generated and then processed by a single serial learner using a global replay database. Alternatively, a single actor can generate data into a local replay memory, after which multiple learners can process this data in parallel to maximize the effectiveness of their learning. Using the Arcade Learning Environment, GORILA was evaluated on 49 Atari 2600 games. In 25 games, GORILA achieved 75% of the human score or higher.

The A3C (Asynchronous Advantage Actor-Critic) algorithm was proposed by Mnih et al. [100], in which multiple agents generate data in parallel asynchronously, and DNN controllers are optimized through gradient descent asynchronously. Similar to the GORILA, actor-learners were used asynchronously, however, instead of using separate machines and a parameter server, multiple CPU threads were used on a single computer. The cost of communication can be eliminated by keeping learners on the same computer. It is likely that multiple actors and learners explore different aspects of the environment simultaneously. This approach can maximize diversity by utilizing different exploration policies for each actor-learner. As multiple actor-learners use online updates in parallel, the overall changes being made to the parameters are more

likely to be less correlated over time than the changes made by a single agent. As a result of parallelization, the data is also diverse and decorated, which provides a more practical alternative to experience replay.

IMPALA (Importance Weighted Actor-Learner Architecture) is a scalable DDR learning algorithm proposed by Espeholt et al. [101]. IMPALA uses GPUs and distributed deep learning methods to update mini-batches of data in parallel, which allows it to train large neural networks efficiently with a distributed set of learners uses synchronized parameter updating and is capable of training. In IMPALA, there is a distribution of parameters across the learners and actors retrieve the parameters from all the learners simultaneously while only sending observations to one learner. IMPALA outperforms A3C-based agents on DMLab-30, achieving a 49.4% vs. 23.8% human normalized score.

Heess et al. [102] introduced Distributed Proximal Policy Optimization (DPPO), a DRL approach based on the principle of proximal policy optimization (PPO) [103]. In DPPO, the collection of data and the calculation of gradients are distributed among the workers. The experiments have been conducted with both synchronous and asynchronous updates, and the results have shown that averaging gradients and applying them synchronously leads to better results.

Ape-X [104] is a distributed architecture for DRL that decouples acting from learning. In Ape-X, a shared neural network is used to select actions by actors and the resulting experience is stored in a shared experience replay memory. The neural network is updated by replaying samples of experience, with prioritizing given to the most significant data generated by the actors.

SEED (Scalable, Efficient Deep-RL) was proposed by Espeholt et al. [105]. SEED RL utilizes modern accelerators to improve the speed and efficiency of DRL. SEED RL uses three types of threads: the inference thread, data prefetching threads, and training threads. The inference thread receives a batch of observations, rewards, and episode termination flags, while data prefetching threads sample data as it is added to a FIFO queue or replay buffer. For each of the TPU cores participating in training, the trajectories are pushed to a device buffer. In comparison with the baseline IMPALA, SEED improves the speed by 1.6x (with 2 cores), 4.1 $\times$ (8 cores), and 4.5x (if the batch size is increased linearly with 5 cores).

Acme [106] is a research framework that helps with algorithm development. Acme aims to increase reproducibility in reinforcement learning and simplify the development of novel and creative algorithms. Acme's main advantage is that it can be used to implement large-scale distributed reinforcement learning algorithms enabling operation at enormous scales while maintaining the inherent readability of the code. In most cases, algorithms implemented with Acme result in a distributed agent with a number of separate (parallel) acting, learning, diagnostic, and helper processes. Acme's main design decision, however, is to reuse the same components across simple, single-process implementations and large-scale distributed systems.

In a recent paper, Dai et al. [107] proposed a "hybrid near-on policy" DRL framework, called Coknight, which leverages a game theory-based DNN partition approach to achieve fast and dynamic partitioning in distributed DRL architectures.

Table 4 provides an overview of these algorithms.

Table 4 Distributer DRL

Algorithm	Articles	Year	No. of references	Simulation/dataset	Evaluation metrics
Distributed deep reinforcement learning	[107]	2022	42	• Atari games	• Onvergence rate • Convergence time • Running time • GPU usage • Memory usage • Bandwidth consumption
	[106]	2020	127	• 5 Atari games: Asterix, • Breakout, MsPacman, Pong and SpaceInvaders • Arcade Learning Environment • DeepMind Control suite • Gym environments	• Mean and standard deviation • Speed
	[105]	2019	53	• Atari-57 • DeepMind Lab • Google Research Football	• Training cost • Speed
	[101]	2018	41	• Atari-57 • DMLab-30	• Median and Mean Human-Normalized scores
	[104]	2018	40	• Atari games	• Median and Mean Human-Normalized scores
	[100]	2016	43	• Atari games • TORCS 3D • Mujoco • Labyrinth	• Median and Mean Human-Normalized scores
	[98]	2015	19	• 49 games from Atari 2600 games	• Human Score

Conclusions and research directions

Distributed machine learning is becoming increasingly important due to the increase in data, the need for more accurate models, the ability to solve complex problems, and the reduction of computation time.

Researchers have proposed different methods for distributing machine learning algorithms, including distributed algorithms for classification, clustering, deep learning, and reinforcement learning. In the case of traditional machine learning methods (clustering and classification algorithms), some studies have attempted to develop distributed versions of them. Our study reviewed the distribution of Boosting, SVM, Consensus-based, and K-means algorithms. For deep learning, there are four types of parallelism: data parallelism, model parallelism, pipelining parallelism, and hybrid parallelism. The majority of these studies considered neural networks such as ResNet, VGG, and AlexNet. In the case of reinforcement learning, researchers have proposed various distributed reinforcement learning algorithms, including A3C, IMPALA, DPPO, Ape-X, SEED RL, and Acme. Distributed machine learning has several limitations that need to be addressed in future research. These limitations include:

- Lack of attention to distributed traditional machine learning: There has been a significant focus on distributed deep learning in recent studies and less attention has been paid to distributed traditional machine learning. Although machine learning algorithms have their advantages and have shown promising results in a number

of areas, they have not been studied as extensively as deep learning in distributed systems.

- Lack of benchmarks: Most studies used MNIST and ImageNet datasets to evaluate their proposed method, but there is no benchmarking to evaluate and compare the performance of existing approaches. Researchers considered a wide range of models, datasets, and evaluation metrics, and even in distributed RL, each study evaluated its method on different types of Atari games. Consequently, benchmarks are necessary to compare the results of different methods.
- Interpretability: Even though DNNs have excellent performance in many areas, understanding their results, particularly in distributed systems, can be challenging. A model's interpretability can help to provide insight into the relationship between input data and the trained model, which is particularly useful in critical domains like healthcare. The interpretability of distributed algorithms remains an open problem.
- New issues: New subjects arise when we try to have distributed algorithms, including the way data and model are partitioned, optimality, delay of the slowest node, communication overhead, scalability, and aggregation of results. These issues need to be addressed to succeed at distributed training and to make it more accessible to data scientists and researchers.

Therefore, this is an open line of research that will have to overcome these new challenges in the future.

Abbreviations

AI	Artificial intelligence
BSP	Bulk synchronous parallel
DNN	Deep neural network
DRL	Deep reinforcement learning
IOT	Internet of Things
ML	Machine learning
NLP	Natural language processing
PPO	Proximal policy optimization
SVM	Support vector machines
SGD	Stochastic gradient descent
S-SGD	Synchronous stochastic gradient descent
WSNs	Wireless sensor networks

Acknowledgements
Not applicable.

Author contributions
MD and ZY wrote the paper .All authors discussed the results, commented on the manuscript, reviewed drafts of the article, and approved the final draft.

Funding
No Funding.

Availability of data and materials
No data were used to support this study.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 14 July 2023 Accepted: 17 September 2023
Published online: 13 October 2023

References

1. Sarker IH, Furhad MH, Nowrozy R. Ai-driven cybersecurity: an overview, security intelligence modeling and research directions. SN Comput Sci. 2021;2:1–18.
2. Duan L, Da Xu L. Business intelligence for enterprise systems: a survey. IEEE Trans Industr Inf. 2012;8(3):679–87.
3. Zhang C, Lu Y. Study on artificial intelligence: the state of the art and future prospects. J Industrial Inform Integr. 2021;23: 100224.
4. Alloghani M, Al-Jumeily D, Mustafina J, Hussain A, Aljaaf AJ. A systematic review on supervised and unsupervised machine learning algorithms for data science. In: Berry MW, Mohamed A, Yap BW, editors. Supervised and unsupervised learning for data science. Cham: Springer International Publishing; 2020. p. 3–21.
5. Sandhu T. Machine learning and natural language processing – a review. Int J Adv Res Comput Sci. 2018;9:582–4.
6. Witten IH, Frank E. Data mining: practical machine learning tools and techniques with java implementations. ACM SIGMOD Rec. 2002;31(1):76–7.
7. Han JM. Kamber in J. Pei, Data mining: concepts and techniques: concepts and techniques, Amsterdam, Elsevier; 2011:3
8. Janiesch C, Zschech P, Heinrich K. Machine learning and deep learning. Electron Markets. 2021;31(3):685–95.
9. Dong S, Wang P, Abbas K. A survey on deep learning and its applications. Comput Sci Rev. 2021;40: 100379.
10. Sarker IH. Machine learning: algorithms, real-world applications and research directions. SN Comput Sci. 2021;2(3):160.
11. Sarıgül M, Karacan L. Region contrastive camera localization. Pattern Recognit Lett. 2023;169:110–7.
12. Sarıgül M. A survey on digital video stabilization. Multimedia Tools Appl. 2023;1:1–27.
13. Liu P, Yuan W, Fu J, Jiang Z, Hayashi H, Neubig G. Pre-train, prompt, and predict: a systematic survey of prompting methods in natural language processing. ACM-CSUR. 2023;55(9):1–35.
14. Munir H, Vogel B, Jacobsson A. Artificial intelligence and machine learning approaches in digital education: a systematic revision. Information. 2022;13(4): 203.
15. Reddy S, Allan S, Coghlan S, Cooper P. A governance model for the application of AI in health care. J Am Med Inform Assoc. 2020;27(3):491–7.
16. Ngai EW, Wu Y. Machine learning in marketing: a literature review, conceptual framework, and research agenda. J Bus Res. 2022;145:35–48.
17. Megnidio-Tchoukouegno M, Adedeji JA. Machine learning for road traffic accident improvement and environmental resource management in the transportation sector. Sustainability. 2023;15(3): 2014.
18. Entezari A, Aslani A, Zahedi R, Noorollahi Y. Artificial intelligence and machine learning in energy systems: a bibliographic perspective. Energ Strat Rev. 2023;45: 101017.
19. Ihme M, Chung WT, Mishra AA. Combustion machine learning: principles, progress and prospects. Prog Energy Combust Sci. 2022;91: 101010.
20. Lee J, Davari H, Singh J, Pandhare V. Industrial artificial intelligence for industry 4.0-based manufacturing systems. Manuf Lett. 2018;18:20–3.
21. Elshawi R, Maher M, Sakr S. Automated machine learning: state-of-the-art and open challenges. arXiv preprint arXiv:190602287. 2019.
22. Xing EP, Ho Q, Xie P, Wei D. Strategies and principles of distributed machine learning on big data. Engineering. 2016;2(2):179–95.
23. Peteiro-Barral D, Guijarro-Berdiñas B. A survey of methods for distributed machine learning. Progress in Artificial Intelligence. 2013;2:1–11.
24. Khalid N, Qayyum A, Bilal M, Al-Fuqaha A, Qadir J. Privacy-preserving artificial intelligence in healthcare: techniques and applications. Comput Biol Med. 2023;158: 106848.
25. Agarwal A, Duchi JC. Distributed delayed stochastic optimization. In: Proceedings of the 24th international conference on neural information processing systems; Granada, Spain: Curran Associates Inc; 2011. p. 873–81.
26. Niu F, Recht B, Re C, Wright SJ. HOGWILD! a lock-free approach to parallelizing stochastic gradient descent. In: Proceedings of the 24th international conference on neural information processing systems; Granada, Spain. Curran Associates Inc; 2011. p. 693–701.
27. Distributed deep neural networks over the cloud, the edge and end devices. In: Teerapittayanon S, McDanel B, Kung H-T, editors. 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE; 2017.
28. Verbraeken J, Wolting M, Katzy J, Kloppenburg J, Verbelen T, Rellermeyer JS. A survey on distributed machine learning. Acm Comput Surv (csur). 2020;53(2):1–33.
29. Langer M, He Z, Rahayu W, Xue Y. Distributed training of deep learning models: a taxonomic perspective. IEEE Trans Parallel Distrib Syst. 2020;31(12):2802–18.
30. Ouyang S, Dong D, Xu Y, Xiao L. Communication optimization strategies for distributed deep neural network training: a survey. J Parallel Distrib Comput. 2021;149:52–65.
31. Tang Z, Shi S, Chu X, Wang W, Li B. Communication-efficient distributed deep learning: a comprehensive survey. arXiv Preprint arXiv:200306307. 2020.
32. Nassef O, Sun W, Purmehdi H, Tatipamula M, Mahmoodi T. A survey: distributed machine learning for 5G and beyond. Comput Netw. 2022;207: 108820.
33. Mayer R, Jacobsen H-A. Scalable deep learning on distributed infrastructures: challenges, techniques, and tools. ACM Comput Surv (CSUR). 2020;53(1):1–37.
34. Muscinelli E, Shinde SS, Tarchi D. Overview of distributed machine learning techniques for 6G networks. Algorithms. 2022;15(6): 210.

35. Yin Q, Yu T, Shen S, Yang J, Zhao M, Huang K et al. Distributed deep reinforcement learning: a survey and a multi-player multi-agent learning toolbox. arXiv preprint arXiv:221200253. 2022.
36. Antwi-Boasiako E, Zhou S, Liao Y, Liu Q, Wang Y, Owusu-Agyemang K. Privacy preservation in distributed deep learning: a survey on distributed deep learning, privacy preservation techniques used and interesting research directions. J Inform Secur Appl. 2021;61:102949.
37. Supervised classification algorithms in machine learning: a survey and review. In: Sen PC, Hajra M, Ghosh M, editors. Emerging Technology in Modelling and Graphics: Proceedings of IEM Graph 2018. Springer; 2020
38. Ferreira AJ, Figueiredo MAT. Boosting algorithms: a review of methods, theory, and applications. In: Zhang C, Ma Y, editors. Ensemble machine learning: methods and applications. New York: Springer; 2012. p. 35–85.
39. In: Freund Y, Schapire RE, editors. Experiments with a new boosting algorithm. Citeseer; 1996. icml.
40. Lazarevic A, Obradovic Z. Boosting algorithms for parallel and distributed learning. Distrib Parallel Databases. 2002;11:203–29.
41. A Resilient Distributed Boosting Algorithm. In: Filmus Y, Mehalel I, Moran S, editors. International Conference on Machine Learning. PMLR; 2022.
42. Hard-core distributions for somewhat hard problems. In: Impagliazzo R, editor. Proceedings of IEEE 36th Annual Foundations of Computer Science. IEEE; 1995.
43. Distributed boosting algorithm for classification of text documents. In: Sarnovsky M, Vronc M, editors. 2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI). IEEE; 2014.
44. Cooper J, Reyzin L, editors. Improved algorithms for distributed boosting. 2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton). 2017: IEEE.
45. Sheykhmousa M, Mahdianpari M, Ghanbari H, Mohammadimanesh F, Ghamisi P, Homayouni S. Support vector machine versus random forest for remote sensing image classification: a meta-analysis and systematic review. IEEE J Sel Top Appl Earth Observations Remote Sens. 2020;13:6308–25.
46. Kuo B-C, Ho H-H, Li C-H, Hung C-C, Taur J-S. A kernel-based feature selection method for SVM with RBF kernel for hyperspectral image classification. IEEE J Sel Top Appl Earth Observations Remote Sens. 2013;7(1):317–26.
47. Dong J-x, Krzyżak A, Suen CY. A fast parallel optimization for training support vector machine. Machine learning and data mining in pattern recognition: Third International Conference, MLDM 2003 Leipzig, July 5–7, 2003 Proceedings, Springer.
48. Lu Y, Roychowdhury V, Vandenberghe L. Distributed parallel support vector machines in strongly connected networks. IEEE Trans Neural Networks. 2008;19(7):1167–78.
49. Alham NK, Li M, Liu Y, Hammoud S. A mapreduce-based distributed SVM algorithm for automatic image annotation. Comput Math Appl. 2011;62(7):2801–11.
50. Ke X, Jin H, Xie X, Cao J. A distributed SVM method based on the iterative MapReduce. Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015); 2015: IEEE.
51. Wang L, Liu H, Su H, Wang J. Bathymetry retrieval from optical images with spatially distributed support vector machines. GIScience & Remote Sensing. 2019;56(3):323–37.
52. Rodriguez MZ, Comin CH, Casanova D, Bruno OM, Amancio DR, Costa LF, et al. Clustering algorithms: a comparative approach. PLoS ONE. 2019;14(1): e0210236.
53. Qin J, Fu W, Gao H, Zheng WX. Distributed $ k $-means algorithm and fuzzy $ c $-means algorithm for sensor networks based on multiagent consensus theory. IEEE Trans Cybernetics. 2016;47(3):772–83.
54. Vendramin L, Campello RJGB, Coletta LF, Hruschka ERs. Distributed fuzzy clustering with automatic detection of the number of clusters. International Symposium on Distributed Computing and Artificial Intelligence; 2011: Springer.
55. Li XY, Yu LY, Lei H, Tang XF. The parallel implementation and application of an improved K-means algorithm. Dianzi Keji Daxue Xuebao/Journal of the University of Electronic Science and Technology of China. 2017;46:61–8.
56. Yuan C, Yang H. Research on K-value selection method of K-means clustering algorithm. J. 2019;2(2):226–35.
57. Patel S, Patel V, Jinwala D, editors. Privacy preserving distributed k-means clustering in malicious model using zero knowledge proof. Distributed Computing and Internet Technology: 9th International Conference, ICDCIT 2013, Bhubaneswar, February 5–8, 2013 Proceedings 9; 2013: Springer.
58. Oliva G, Setola R, Hadjicostis CN. Distributed k-means algorithm. arXiv Preprint arXiv:13124176. 2013.
59. Benchara FZ, Youssfi M. A new scalable distributed k-means algorithm based on cloud micro-services for high-performance computing. Parallel Comput. 2021;101: 102736.
60. Datta S, Giannella C, Kargupta H. Approximate distributed k-means clustering over a peer-to-peer network. IEEE Trans Knowl Data Eng. 2008;21(10):1372–88.
61. Ding H, Liu Y, Huang L, Li J. K-means clustering with distributed dimensions. International Conference on Machine Learning; 2016: PMLR.
62. Farkas A, Kertész G, Lovas R. Parallel and distributed training of deep neural networks: a brief overview. 2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES); 2020: IEEE.
63. Schmidhuber J. Deep learning in neural networks: an overview. Neural Netw. 2015;61:85–117.
64. Akintoye SB, Han L, Zhang X, Chen H, Zhang D. A hybrid parallelization approach for distributed and scalable deep learning. IEEE Access. 2022;10:77950–61.
65. Diskin M, Bukhtiyarov A, Ryabinin M, Saulnier L, Sinitsin A, Popov D, et al. Distributed deep learning in open collaborations. Adv Neural Inf Process Syst. 2021;34:7879–97.
66. Ben-Nun T, Hoefler T. Demystifying parallel and distributed deep learning: an in-depth concurrency analysis. ACM Comput Surv (CSUR). 2019;52(4):1–43.
67. Dean J, Corrado GS, Monga R, Chen K, Devin M, Le QV, et al. Large scale distributed deep networks. In: Proceedings of the 25th international conference on neural information processing systems - Vol. 1; Lake Tahoe, Nevada: Curran Associates Inc.; 2012. p. 1223–31.
68. Zhang S, Choromanska A, LeCun Y. Deep learning with elastic averaging SGD. In: Proceedings of the 28th international conference on neural information processing systems - Vol. 1; Montreal, Canada: MIT Press; 2015. p. 685–93.

69. George J, Gurram P. Distributed deep learning with event-triggered communication. arXiv Preprint arXiv:190905020. 2019.
70. Kim S, Yu G-I, Park H, Cho S, Jeong E, Ha H, et al. Parallax: sparsity-aware data parallel training of deep neural networks. Proceedings of the Fourteenth EuroSys Conference 2019; 2019.
71. Ye Q, Zhou Y, Shi M, Sun Y, Lv J. DBS: dynamic batch size for distributed deep neural network training. arXiv Preprint arXiv:200711831. 2020.
72. Song Z, Gu Y, Wang Z, Yu G. DRPS: efficient disk-resident parameter servers for distributed machine learning. Front Comput Sci. 2022;16:1–12.
73. Sergeev A, Del Balso M. Horovod: fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv:180205799. 2018.
74. Alqahtani S, Demirbas M. Performance analysis and comparison of distributed machine learning systems. arXiv Preprint arXiv:190902061. 2019.
75. Horvóth S, Ho C-Y, Horvath L, Sahu AN, Canini M, Richtárik P. Natural compression for distributed deep learning. Mathematical and scientific machine learning: PMLR; 2022.
76. Jia Z, Zaharia M, Aiken A. Beyond data and model parallelism for deep neural networks. Proceedings of Machine Learning and Systems. 2019;1:1–13.
77. Gandhi S, Iyer AP, editors. P3: distributed deep graph learning at scale. OSDI; 2021.
78. Mirhoseini A, Pham H, Le QV, Steiner B, Larsen R, Zhou Y, et al, editors. Device placement optimization with reinforcement learning. International conference on machine learning; 2017: PMLR.
79. Zhuang H, Wang Y, Liu Q, Lin Z. Fully decoupled neural network learning using delayed gradients. IEEE Trans Neural Networks Learn Syst. 2021;33(10):6013–20.
80. Huo Z, Gu B, Huang H. Decoupled parallel backpropagation with convergence guarantee. International Conference on Machine Learning; 2018: PMLR.
81. Kim C, Lee H, Jeong M, Baek W, Yoon B, Kim I et al.: Torchgpipe: On-the-fly pipeline parallelism for training giant models. arXiv Preprint arXiv:200409910. 2020.
82. Narayanan D, Harlap A, Phanishayee A, Seshadri V, Devanur NR, Ganger GR, et al, editors. PipeDream: generalized pipeline parallelism for DNN training. Proceedings of the 27th ACM symposium on operating systems principles; 2019.
83. Chen C-C, Yang C-L, Cheng H-Y. Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. arXiv preprint arXiv:180902839. 2018.
84. Lee S, Jha D, Agrawal A, Choudhary A, Liao W-K, editors. Parallel deep convolutional neural network training by exploiting the overlapping of computation and communication. 2017 IEEE 24th international conference on high performance computing (HiPC); 2017: IEEE.
85. Wang M, Huang C-c, Li J.: Unifying data, model and hybrid parallelism in deep learning via tensor tiling. arXiv preprint arXiv:180504170. 2018.
86. Ono J, Utiyama M, Sumita E. Hybrid data-model parallel training for sequence-to-sequence recurrent neural network machine translation. arXiv Preprint arXiv:190900562. 2019.
87. Song L, Mao J, Zhuo Y, Qian X, Li H, Chen Y, editors. Hypar: towards hybrid parallelism for deep learning accelerator array. 2019 IEEE international symposium on high performance computer architecture (HPCA); 2019: IEEE.
88. Liu J, Wu Z, Feng D, Zhang M, Wu X, Yao X, et al. Heterps: distributed deep learning with reinforcement learning based scheduling in heterogeneous environments. Fut Generat Computer Syst. 2023. https://doi.org/10.1016/j.future.2023.05.032.
89. Yadan O, Adams K, Taigman Y, Ranzato MA. Multi-gpu training of convnets. arXiv Preprint arXiv:13125853. 2013.
90. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Commun ACM. 2017;60(6):84–90.
91. Oyama Y, Maruyama N, Dryden N, McCarthy E, Harrington P, Balewski J, et al. The case for strong scaling in deep learning: training large 3d cnns with hybrid parallelism. IEEE Trans Parallel Distrib Syst. 2020;32(7):1641–52.
92. Povey D, Zhang X, Khudanpur S. Parallel training of DNNs with natural gradient and parameter averaging. arXiv Preprint arXiv:14107455. 2014.
93. Yu H, Yang S, Zhu S, editors. Parallel restarted SGD with faster convergence and less communication: demystifying why model averaging works for deep learning. Proceedings of the AAAI conference on artificial intelligence; 2019.
94. Cheatham T, Fahmy A, Stefanescu D, Valiant L. Bulks ynchronous parallel computing—a paradigm for transportable software. In: Zaky A, Lewis T, editors. Tools and environments for parallel and distributed systems. Berlin: Springer; 1996.
95. Mousavi SS, Schukat M, Howley E. Deep reinforcement learning: an overview. Proceedings of SAI intelligent systems conference (IntelliSys) 2016:2;2018 Springer.
96. Le N, Rathour VS, Yamazaki K, Luu K, Savvides M. Deep reinforcement learning in computer vision: a comprehensive survey. Artif Intell Rev. 2022:1–87.
97. Samsami MR, Alimadad H. Distributed deep reinforcement learning: an overview. arXiv preprint arXiv:201111012. 2020.
98. Nair A, Srinivasan P, Blackwell S, Alcicek C, Fearon R, De Maria A, et al. Massively parallel methods for deep reinforcement learning. arXiv Preprint arXiv:150704296. 2015.
99. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D et al. Playing atari with deep reinforcement learning. arXiv Preprint arXiv:13125602. 2013.
100. Asynchronous methods for deep reinforcement learning. In: Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, et al. editors. International Conference on Machine Learning. PMLR; 2016.
101. Espeholt L, Soyer H, Munos R, Simonyan K, Mnih V, Ward T, et al. editors. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. International Conference on Machine Learning. PMLR; 2018.
102. Heess N, TB D, Sriram S, Lemmon J, Merel J, Wayne G et al. Emergence of locomotion behaviours in rich environments. arXiv Preprint arXiv:170702286. 2017.

103. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv Preprint arXiv:170706347. 2017.
104. Horgan D, Quan J, Budden D, Barth-Maron G, Hessel M, Van Hasselt H et al. Distributed prioritized experience replay. arXiv preprint arXiv:180300933. 2018.
105. Espeholt L, Marinier R, Stanczyk P, Wang K, Michalski M. Seed rl: scalable and efficient deep-rl with accelerated central inference. arXiv Preprint arXiv:191006591. 2019.
106. Hoffman MW, Shahriari B, Aslanides J, Barth-Maron G, Momchev N, Sinopalnikov D et al. Acme: a research framework for distributed reinforcement learning. arXiv Preprint arXiv:200600979. 2020.
107. Dai H, Wu J, Wang Y, Xu C. Towards scalable and efficient deep-RL in edge computing: a game-based partition approach. J Parallel Distrib Comput. 2022;168:108–19.

Publisher's Note