A Survey of On-Device Machine Learning: An Algorithms and Learning Theory Perspective

SAUPTIK DHAR, JUNYAO GUO, JIAYI (JASON) LIU, SAMARTH TRIPATHI, UNMESH KURUP, and MOHAK SHAH, America Research Center, LG Electronics

The predominant paradigm for using machine learning models on a device is to train a model in the cloud and perform inference using the trained model on the device. However, with increasing numbers of smart devices and improved hardware, there is interest in performing model training on the device. Given this surge in interest, a comprehensive survey of the field from a device-agnostic perspective sets the stage for both understanding the state of the art and for identifying open challenges and future avenues of research. However, on-device learning is an expansive field with connections to a large number of related topics in AI and machine learning (including online learning, model adaptation, one/few-shot learning, etc.). Hence, covering such a large number of topics in a single survey is impractical. This survey finds a middle ground by reformulating the problem of on-device learning as resource constrained learning where the resources are compute and memory. This reformulation allows tools, techniques, and algorithms from a wide variety of research areas to be compared equitably. In addition to summarizing the state of the art, the survey also identifies a number of challenges and next steps for both the algorithmic and theoretical aspects of on-device learning.

CCS Concepts: • **Theory of computation** → **Sample complexity and generalization bounds**; **Design and analysis of algorithms**; *Query learning*; • **Hardware** → *Emerging tools and methodologies*; • **Computing methodologies** → **Machine learning**; *Computer vision*; • **Mathematics of computing**;

Additional Key Words and Phrases: On-device learning, machine learning, algorithm development

ACM Reference format:
Sauptik Dhar, Junyao Guo, Jiayi (Jason) Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. 2021. A Survey of On-Device Machine Learning: An Algorithms and Learning Theory Perspective. *ACM Trans. Internet Things* 2, 3, Article 15 (July 2021), 45 pages.
https://doi.org/10.1145/3450494

1 INTRODUCTION

The addition of intelligence to a device carries the promise of a seamless experience that is tailored to each user's specific needs while maintaining the integrity of their personal data. The current approach to making such intelligent devices is based on a cloud paradigm where data are collected at the device level and transferred to the cloud. Once transferred, these data are then aggregated with data collected from other devices, processed, and used to train a machine learning model.

Authors' addresses: S. Dhar, J. Guo, J. (Jason) Liu, S. Tripathi, U. Kurup, and M. Shah, America Research Center, LG Electronics 5150 Great America Pkwy, Santa Clara, CA 95054; emails: {sauptik.dhar, junyao.guo, jason.liu, samarth.tripathi}@lge.com, unmesh@ukurup.com, mohak@mohakshah.com.

2577-6207/2021/07-ART15 $15.00
https://doi.org/10.1145/3450494

When the training is done, the resulting model is pushed from the cloud back to the device where it is used to improve the device's intelligent behavior. In the cloud paradigm, all machine learning that happens on the device is inference, that is, the execution of a model that was trained in the cloud. This separation of roles—data collection and inference on the edge, data processing and model training in the cloud—is natural given that end-user devices have form-factor and cost considerations that impose limits on the amount of computing power and memory they support, as well as the energy that they consume.

Cloud-based systems have access to nearly limitless resources and are constrained only by cost considerations making them ideal for resource intensive tasks like data storage, data processing, and model building. However, the cloud-based paradigm also has drawbacks that will become more pronounced as AI becomes an ubiquitous aspect of consumer life. The primary considerations are in the privacy and security of user data, as these data need to be transmitted to the cloud and stored there, most often, indefinitely. Transmission of user data is open to interference and capture, and stored data leaves open the possibility of unauthorized access.

In addition to privacy and security concerns, the expectation for intelligent devices will be that their behavior is tailored specifically to each consumer. However, cloud-trained models are typically less personalized, as they are built from data aggregated from many consumers, and each model is built to target broad user segments, because building individual models for every consumer and every device is cost prohibitive in most cases. This de-personalization also applies to distributed paradigms like federated learning that typically tend to improve a global model based on averaging the individual models [128].

pt Finally, AI-enabled devices will also be expected to learn and respond instantly to new scenarios but cloud-based training is slow, because added time is needed to transmit data and models back and forth from the device. Currently, most use cases do not require real-time model updates, and long delays between data collection and model updates is not a serious drawback. But, as intelligent behavior becomes commonplace and expected, there will be a need for real-time updates, like in the case of connected vehicles and autonomous driving. In such situations, long latency becomes untenable and there is a need for solutions where model updates happen locally and not in the cloud.

As devices become more powerful, it becomes possible to address the drawbacks of the cloud model by moving some or all of the model development onto the device itself. Model training, especially in the age of deep learning, is often the most time-consuming part of the model development process, making it the obvious area of focus to speed up model development on the device. Doing model training on the device is often referred to variously as *Learning on the Edge* and *On-device Learning*. However, we distinguish between these terms, with *learning on the edge* used as a broad concept to signify the capability of real or quasi-real-time learning without uploading data to the cloud while *on-device learning* refers specifically to the concept of doing model training on the resource-constrained device itself.

1.1 On-device Learning

1.1.1 Definition of an Edge Device. Before we elaborate on on-device learning, it is helpful to define what we mean by a device, or specifically an edge device, in the context of on-device learning. We define an edge device to be a device whose compute, memory, and energy resources are constrained and cannot be easily increased or decreased. These constraints may be due to form-factor considerations (it is not possible to add more compute or memory or battery without increasing the size of the device) or due to cost considerations (there is enough space to add a GPU to a washing machine but this would increase its cost prohibitively). This definition of an edge device applies to all such consumer and industrial devices where resource constraints place limitations on what is available for building and training AI models. Any cloud solution such as Amazon AWS, Google

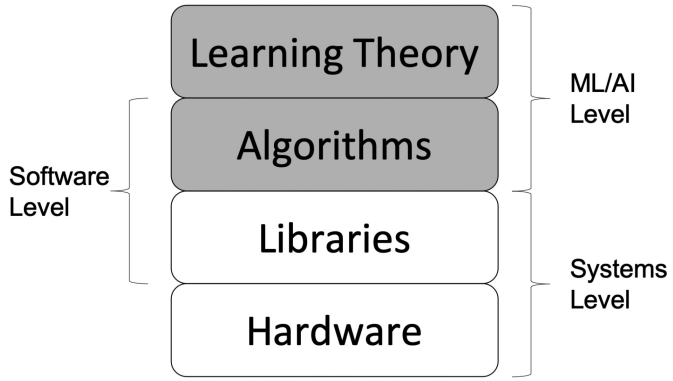

Fig. 1. Levels of constraints on edge devices. Only the topics in gray (Learning Theory and Algorithms) fall under the scope of this survey.

Cloud Platform, Microsoft Azure, or even on-premise computing clusters do not fit the edge definition, because it is easy to provision additional resources as needed. Likewise, a workstation would not be considered an edge device, because it is straightforward to replace its CPU, add more memory, and even add an additional GPU card. A standard laptop, however, would be considered an edge device as it is not easy to add additional resources as needed, even though their resources generally far exceed what is normally considered as available in a consumer edge device.

1.1.2 Training Models on an Edge Device. The primary constraints to training models on-device in a reasonable time-frame is the lack of compute and memory on the device. Speeding up training is possible either by adding more resources to the device or using these resources more effectively or some combination of the two. Figure 1 shows a high-level breakdown of the different levels at which these approaches can be applied. Each level in this hierarchy abstracts implementation details of the level below it and presents an independent interface to the level above it.

(1) **Hardware:** At the bottom of the hierarchy are the actual chipsets that execute all learning algorithms. Fundamental research in this area aims at improving existing chip design (by developing chips with more compute and memory, and lower power consumption and footprint) or developing new designs with novel architectures that speed up model training. While hardware research is a fruitful avenue for improving on-device learning, it is an expensive process that requires large capital expenditure to build laboratories and fabrication facilities, and usually involves long timescales for development.
(2) **Libraries:** Every machine learning algorithm depends on a few key operations (such as Multiply-Add in the case of neural networks). The libraries that support these operations are the interface that separate the hardware from the learning algorithms. This separation allows for algorithm development that is not based on any specific hardware architecture. Improved libraries can support faster execution of algorithms and speed up on-device training. However, these libraries are heavily tuned to the unique aspects of the hardware on which the operations are executed. This dependency limits the amount of improvement that can be gained by new libraries.
(3) **Algorithms:** Since on-device learning techniques are grounded in their algorithmic implementations, research in novel algorithm development is an important part of making model training more efficient. Such algorithm development can take into account resource constraints as part of the model training process. Algorithm development leads to hardware-independent techniques but the actual performance of each algorithm is specific

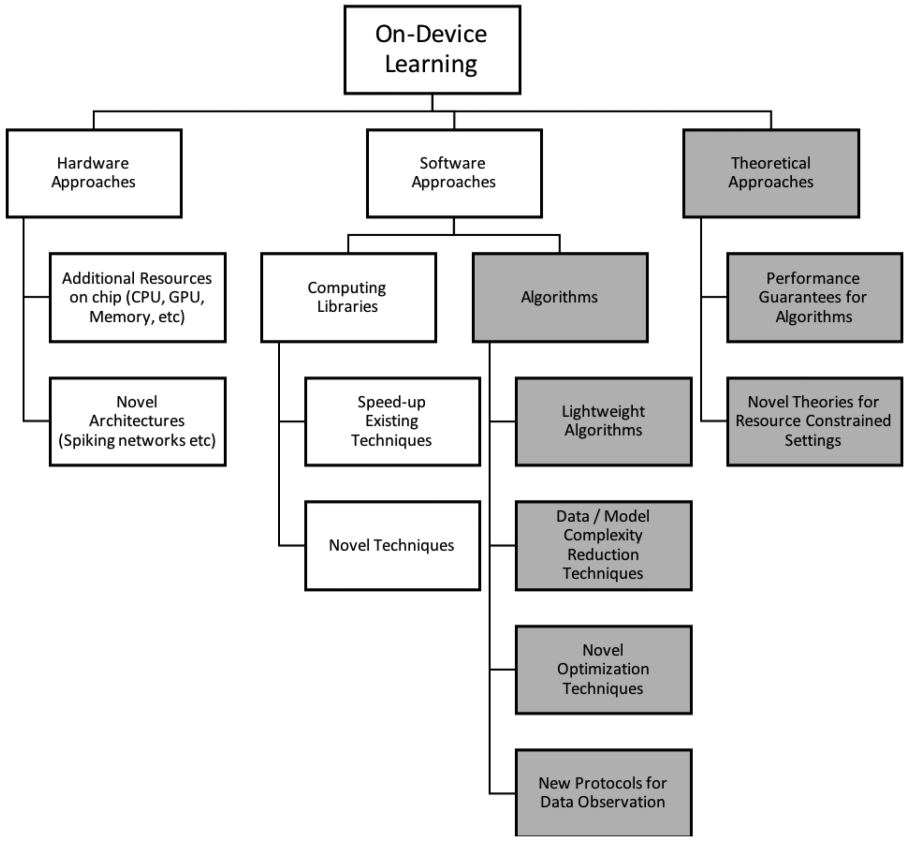

Fig. 2. Different approaches to improving on-device learning. Topics in gray are covered in this survey.

to the exact domain, environment, and hardware, and needs to be verified empirically for each configuration. Depending on the number of choices available in each of these dimensions, the verification space could become very large.

(4) **Theory:** Every learning algorithm is based on an underlying theory that guarantees certain aspects of its performance. Developing novel theories targeted at on-device learning help us understand how algorithms will perform under resource-constrained settings. However, while theoretical research is flexible enough to apply across classes of algorithms and hardware systems, it is limited due to the inherent difficulty of such research and the need to implement a theory in the form of an algorithm before its utility can be realized.

Figure 2 shows an expanded hierarchical view of the different levels of the edge learning stack and highlight different ways to improve the performance of model training on the device at each level. The hardware approaches involve either adding additional resources to the restricted form-factor of the device or developing novel architectures that are more resource efficient. Software approaches to improve model training involve either improving the performance of computing libraries such as OpenBLAS, Cuda, and CuDNN or improving the performance of the machine learning algorithms themselves. Finally, theoretical approaches help direct new research on **Machine Learning (ML)** algorithms and improve our understanding of existing techniques and their generalizability to new problems, environments, and hardware.

1.2 Scope of This Survey

There is a large ongoing research effort, mainly in academia, that looks at on-device learning from multiple points of view including single vs. multiple edge devices, hardware vs. software vs. theory, and domain of application such as healthcare vs. consumer devices vs. autonomous cars. Given the significant amount of research in each of these areas it is important to restrict this survey to a manageable subset that targets the most important aspects of on-device learning.

We first limit this survey to the *Algorithms* and *Learning Theory* levels in Figure 1. This allows us to focus on the machine learning aspects of on-device learning and develop new techniques that are independent of specific hardware. We also limit the scope of this survey to learning on a single device. This restriction makes the scope of the survey manageable while also providing a foundation on which to expanded to distributed settings. In addition, at the right level of abstraction, a distributed edge system can be considered as a single device with an additional resource focused on communication latency. This view allows us to extend single-device algorithms and theories to the distributed framework at a later stage.

The goal of this survey then is to provide a large-scale view of the current state of the art in algorithmic and theoretical advances for on-device learning on single devices. To accomplish this goal, the survey reformulates the problem of on-device learning as one of resource constrained learning. This reformulation describes the efficiency of on-device learning using two resources—compute and memory—and provides a foundation for a fair comparison of different machine learning and AI techniques and their suitability for on-device learning. Finally, this survey identifies challenges in algorithms and theoretical considerations for on-device learning and provides the background needed to develop a road map for future research and development in this field.

1.3 How to Read This Survey

This survey is a comprehensive look at the current state of the art in training models on resource-constrained devices. It is divided into four main sections excluding the introduction and the conclusion. Section **??** briefly introduces the resources and their relevance to on-device learning. Sections 3 and 4, respectively, focus on the algorithmic and theoretical levels of the edge platform hierarchy. Finally Section 5 provides a brief summary and identifies various challenges in making progress toward a robust framework for on-device learning. For those interested in specific aspects of on-device learning, Sections 3 and 4 are mostly self-contained and can be read separately.

Resource Constraints in On-Device Learning (in Section 2): briefly discusses the relevant resources that differentiate on-device learning from a cloud-based system. Most existing research at the various levels in Figure 1 is targeted toward addressing on-device learning when there is limited availability of these resources.

Algorithm Research (in Section 3): addresses recent algorithmic developments toward accurately capturing the hardware constraints in a software framework and then surveys the state of the art in machine learning algorithms that take into account resource constraints. This section categorizes the algorithms from a computational perspective (i.e., the underlying computational model used).

Theory Research (in Section 4): addresses on-device learning from a statistical perspective and surveys traditional learning theories forming the basis of most of the algorithm designs addressed in Section 3. It later addresses the "un-learnability" problem in a resource constrained setting and surveys newer resource constrained learning theories. Such newer theories abstract the resource constraints (i.e., memory, processing speed, etc.) as an information bottleneck and provides performance guarantees for learning under these settings.

Finally, Section 5 summarises the previous sections and addresses some of the open challenges in on-device learning research.

2 RESOURCE CONSTRAINTS IN ON-DEVICE LEARNING

The main difference between traditional machine learning and learning/inference on edge are the additional constraints imposed by device resources. Designing AI capabilities that run on a device necessitates building machine learning algorithms with high model accuracy while concurrently maintaining the resource constraints enforced by the device. This section discusses these critical resource constraints that pose major challenges while designing learning algorithms for edge devices.

2.1 Processing Speed

The response time is often among the most critical factors for the usability of any on-device application [141]. The two commonly used measurements are *throughput* and *latency*. Throughput is measured as the rate at which the input data are processed. To maximize throughput it is common to group inputs into batches resulting in higher utilization. But, measuring this incurs additional wait time for aggregating data into batches. Hence, for time-critical use cases, latency is the more frequently used measure. Latency characterizes the time interval between a single input and its response. Although throughput is the inverse of latency (when batch size is fixed to 1), the runtime of an application may vary dramatically depending on whether computations are optimized for throughput vs. latency [184]. To simplify our discussion, in this survey we use an abstract notion of *runtime* as a proxy for both *throughput* and *latency*.

For the physical system, the processing speed dictates the runtime of an application. This speed is typically measured in clock frequency (i.e., the number of cycles per second) of a processor. Within each cycle, a processor carries out a limited number of operations based on the hardware architecture and the types of the operations. For scientific computations, **Floating point operations per second (FLOPS)** is frequently used to measure the number of floating point operations per second. Another frequently used measure specifically for matrix intensive computations such as those in machine learning algorithms is **multiplier–accumulate** (**MAC**). Thus, besides increasing the clock frequency, efficiently combining multiple operations into a single cycle is also an important topic for improving the processing speed.

On a separate note, the processing speed of a system is also sensitive to the communication latency among the components inside a system. As discussed before, the communication latency aspect is better aligned to the distributed/decentralized computing paradigm for edge learning and has not been addressed in this survey.

2.2 Memory

At the heart of any machine learning algorithm is the availability of data for building the model. The second important resource for building AI-driven on-device applications is memory. The memory of a computing device provides immediate data access as compared to other storage components. However, this speed of data access comes at higher costs. For reasons of cost, most edge devices are designed with limited memory. As such, the memory footprint of edge applications are typically tailored for a specific target device.

Advanced machine learning algorithms often take a significant amount of memory during model building through storage of the model parameters and auxiliary variables, and so on. For instance, even relatively simple image classification models like ResNet-50 can take megabytes of memory space (later shown in Table 3). Therefore, designing a lightweight model is of a key aspect of accomplishing machine learning on the device. A detailed survey of such techniques is covered in Sections 3 and 4.

Table 1. Comparison of Hardware Requirements

Use	Device	Hardware Chip	Computing	Memory	Power
Workstation	NVIDIA DGX-2[1]	16 NVIDIA Tesla V100 **GPU**s	2 **PFLOPS**	512 GB	10 kW
Mobile Phone	Pixel 3[2]	Qualcomm Snapdragon™ 845[3]	727 **GFLOPS**	4 GB	34mW (Snapdragon)
Autonomous Driving	NVDIA DRIVE AGX Xavier[4]	NVIDIA Xavier processor	30 TOPS	16 GB	30 W
Smart home	Amazon Echo[5]	TI DM3725 ARM Cortex-A8	up to 1 GHz	256 MB	4 W (peak)
General IoT	Qualcomm AI Engine[6]	Hexagon 685 and Adreno 615	2.1 TOPS	2-4 GB	1 W
General IoT	Raspberry Pi 3[7]	Broadcom ARM Cortex A53	1.2 GHz	1 GB	0.58 W
General IoT	Arduino Uno Rev3[8]	ATmega328P	16 MHz	2 KB	0.3 W

[1] http://images.nvidia.com/content/pdf/dgx-2-print-datasheet-738070-nvidia-a4-web.pdf.
[2] https://store.google.com/product/pixel_3_specs.
[3] https://www.qualcomm.com/products/snapdragon-845-mobile-platform.
[4] https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/.
[5] https://www.ifixit.com/Teardown/Amazon+Echo+Teardown/33953.
[6] https://www.qualcomm.com/products/vision-intelligence-400-platform.
[7] https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/.
[8] https://store.arduino.cc/usa/arduino-uno-rev3.

Besides the model size, querying the model parameters for processing is both time-consuming and energy intensive [167]. For example, a single **MAC** operation requires three memory reads and one memory write. In the worst case, these reads and write may be on the off-chip memory rather than the on-chip buffer. This would result in a significant throughput bottleneck and cause orders of magnitude higher energy consumption [23].

2.3 Power Consumption and Energy Efficiency

Power consumption is another crucial factor for on-device learning. An energy efficient solution can prolong the battery lifetime and cut maintenance costs. The system *power*, commonly studied in hardware development, is the ratio of energy consumption and time span for a given task. However, it is not a suitable measure for machine learning applications on the edge. First, the power consumption depends on the volume of computation required, e.g., data throughput. Second, the application is often capped at the maximum power of the device when the learning task is intensive. Therefore to better quantify the power consumption, the total energy consumption along with the throughput is recommended for comparing energy efficiency.

Linking energy consumption of a particular AI-driven application for a specific device jointly depends on a number of factors like runtime, memory, and so on. Capturing these dependencies are almost never deterministic. Hence, most existing research estimate the power/energy usage through a surrogate function that typically depends on the memory and runtime of an application. A more detailed survey of such advanced approaches are covered in Section 3.

2.4 Typical Use Case and Hardware

Finally, we conclude this section by providing a brief landscape of the variety of edge devices used in several use-case domains and their resource characterization. As seen in Table 1, learning on edge spans a wide spectrum of hardware specifications. Hence, designing machine learning models for edge devices requires a very good understanding of the resource constraints, and appropriately incorporating these constraints into the systems, algorithms, and theoretical design levels.

3 ALGORITHMS FOR ON-DEVICE LEARNING

The algorithms approach targets developing resource-efficient techniques that work with existing resource-constrained platforms. This section provides a detailed survey on the various approaches

to analyze and estimate a ML model's resource footprint and the state-of-the-art algorithms proposed for on-device learning.

We present these algorithms and model optimization approaches in a task-agnostic manner. This section discusses general approaches that adapt both traditional ML algorithms and deep learning models to a resource constrained setting. These approaches can be applied to multiple tasks such as classification, detection, regression, image segmentation, and super-resolution.

Many traditional ML algorithms, such as **Support Vector Machines (SVM)** and Random Forest, are already suitable for multiple tasks and do not need special consideration per task. Deep learning approaches, however, do vary considerably from task to task. However, for the tasks mentioned above, these networks generally deploy a CNN or RNN as the backbone network for feature extraction [91, 130]. As a consequence the resource footprint of training the backbone networks would directly affect the training performance of the overall model. For example, deep learning-based image segmentation methods usually use ResNet as the backbone, while adding additional modules such as a graphical models, modifications to the convolution computation, or combining backbone CNNs with other architectures such as encoder-decoders to achieve their goal [130].

Given these commonalities, we expect the approaches presented in this section to be generalizable to different tasks and refer the readers to surveys [51, 91, 130, 191] for detailed comparison of task-specific models. More importantly, we categorize the directions in which improvements can be made, which could serve as a guideline for proposing novel resource-efficient techniques for new models/tasks. Note that for CNN benchmarking, we use the image classification task as an example as benchmarking datasets are well established for this area. Tasks such as scene segmentation and super-resolution currently lack standard benchmarks making it difficult to equitably compare models.

3.1 Resource Footprint Characterization

Before adapting ML algorithms to the resource-constrained setting, it is important to first understand the resource requirements of common algorithms and identify their resource bottlenecks. The conventional approach adopts an asymptotic analysis (such as the Big-O notation) of the algorithm's implementation. For DNNs, an alternate analysis technique is to use hardware-agnostic metrics such as the number of parameters and number of operations (FLOPs/MACs). However, it has been demonstrated that hardware-agnostic metrics are too crude to predict the real performance of algorithms [120, 124, 203], because these metrics depend heavily on the specific platform and framework used. This hardware dependency has lead to a number of efforts aimed at measuring the true resource requirements of many algorithms (mostly DNNs) on specific platforms. A third approach to profiling proposes building regression models that accurately estimate and predict resource consumption of DNNs from their weights and operations. We will provide an overview of all these resource characterization approaches in this subsection as well as new performance metrics for algorithm analysis that incorporates the algorithm's resource footprint.

3.1.1 Asymptotic Analysis. In this section, we present a comparative overview of the computational and space complexities of both traditional machine learning algorithms and DNNs using asymptotic analysis and hardware-agnostic metrics.

Traditional Machine Learning Algorithms: Due to the heterogeneity of model architectures and optimization techniques, there is no unified approach that characterizes the resource utilization performance of traditional machine learning algorithms. The most commonly used method is asymptotic analysis that quantifies computational complexity and space complexity using the Big-O notation. Table 2 summarizes the computational and space complexities of 10 popular machine learning algorithms based on their implementation in Map-Reduce [27] and Scikit-Learn

Table 2. Comparison of Traditional Machine Learning Algorithms

Algorithm	Model size	Optimization	Training complexity	Inference complexity
Decision tree	$O(m)$	–	$O(mnlog(m))$	$O(log(m))$
Random forest	$O(N_{tree}m)$	–	$O(N_{tree}mnlog(m))$	$O(N_{tree}log(m))$
SVM	$O(n)$	gradient descent	$O(m^2n)$	$O(m_{sv}n)$
Logistic regression	$O(n)$	Newton-Raphson	$O(mn^2+n^3)$	$O(n)$
kNN	$O(mn)$	–	–	$O(mn)$
Naive Bayes	$O(nc)$	–	$O(mn+nc)$	$O(nc)$
Linear regression	$O(n)$	matrix inversion	$O(mn^2+n^3)$	$O(n)$
k-means	–	–	$O(mnc)$	–
EM	–	–	$O(mn^2+n^3)$	–
PCA	–	eigen-decomposition	$O(mn^2+n^3)$	–

Notation: m, number of training samples; n, input dimension; c, number of classes.

[104]. Note that algorithm complexity can vary across implementations, but we believe the results demonstrated in Table 2 are representative of the current landscape. For methods that require iterative optimization algorithms or training steps, the training complexity is estimated for one iteration.

Some key observations can be made from Table 2. First, most traditional machine learning algorithms (except tree-based methods and **k-Nearest Neighbor (kNN)**) have a model size that is linear to the input dimension, which do not require much memory (compared to DNNs, that will be discussed later). Second, except for kNN, which requires distance calculation between the test data and all training samples, the inference step of other algorithms is generally very fast. Third, some methods require complex matrix operations such as matrix inversion and eigen-decomposition with computational complexity around $O(n^3)$ [27]. Therefore, one should consider whether these matrix operations can be efficiently supported by the targeted platform when deploying these methods on resource-constrained devices.

In terms of accuracy of traditional machine learning algorithms, empirical studies have been carried out using multiple datasets [2, 50, 209]. However, for on-device learning, there are few studies that analyze both accuracy and complexity of ML algorithms and the tradeoffs therein.

Deep Neural Networks: Deep neural networks have shown superior performance in many computer vision and natural language processing tasks. To harvest their benefits for edge learning, one has to first evaluate whether the models and the number of operations required can fit into the available resources on a given edge device. Accuracy alone cannot justify whether a DNN is suitable for on-device deployment. To this end, recent studies that propose new DNN architectures also consider MACs/FLOPs and number of weights to provide a qualitative estimate of the model size and computational complexity. However, both memory usage and energy consumption also depend on the feature map or activations [23, 120]. Therefore, in the following, we review popular DNNs in terms of accuracy, weights, activations, and MACs. Accuracies of models are excerpted from best accuracies reported on open platforms and the papers that proposed these models. Number of weights, activations and MACs are either excerpted from papers that first proposed the model or calculated using the Netscope tool [35]. Note that these papers only count the MACs involved in a forward pass. Since the majority of the computational load in training DNNs happens in the backward pass, the values reported do not give a realistic picture of the computational complexity for model training.

Compared to other network architectures, most on-device learning research focuses on CNNs, which can be structured as an acyclic graph and analyzed layer-by-layer. Specifically, a CNN consists of two types of layers, namely, the **convolutional layer (CONV)** and the **fully connected layer (FC)**. It has been shown in Reference [23] that the resource requirement of these two layers

Table 3. Comparison of Popular CNNs

Metric	AlexNet [97]	VGG-16 [158]	GoogLeNet [168]	ResNet-18 [70]	ResNet-50 [70]	Inception v3 [169]
Top-1 acc.	57.2	71.5	69.8	69.6	76.0	76.9
Top-5 acc.	80.2	91.3	90.0	89.2	93.0	93.7
Input size	227×227	224×224	224×224	224×224	224×224	299×299
# of stacked CONV layers	5	13	21	17	49	16
Weights	2.3M	14.7M	6.0M	9.5M	23.6M	22M
Activations	0.94M	15.23M	6.8M	3.2M	11.5M	10.6M
MACs	666M	15.3G	1.43G	1.8G	3.9G	3.8G
# of FC layers	3	3	1	1	1	1
Weights	58.7M	125M	1M	0.5M	2M	2M
Activations	9K	9K	2K	1.5K	3K	3K
MACs	58.7M	125M	1M	0.5M	2M	2M
Total weights	61M	138M	7M	10M	25.6M	24M
Total activations	0.95M	15.24M	6.8M	3.2M	11.5M	10.6M
Total MACs	724M	15.5G	1.43G	1.8G	3.9G	3.8G

can be very different due to their different dataflow and data reuse patterns. In Table 3, we summarize the layerwise statistics of popular high-performance CNN models submitted to the ImageNet challenge. In general, these models are very computation and memory intensive, especially during training. Training requires allocating memory to weights, activations, gradients, data batches, and workspace, which is at least hundreds of MBs if not GBs. These models are hard to deploy on a resource-constrained device, let alone be trained on one. To enable CNN deployment on edge devices, models with smaller sizes and more compact architectures are proposed, which we will review in Section 3.2.2.

3.1.2 Resource Profiling. The most accurate way to quantify the resource requirements of machine learning algorithms is to measure them during deployment. Table 4 summarizes current efforts on DNN benchmarking using various platforms and frameworks. For inference, we only present benchmarks that use at least one edge device such as a mobile phone or an embedded computing system (such as NVidia Jetson TX1). However, as it has not been feasible to train DNNs at large-scale on edge devices yet, we present training benchmarks utilizing single or multiple high-performance CPU or GPUs. Interestingly, apart from measuring model-level performance, three benchmarks [1, 53, 147] further decompose the operations involved in running DNNs and profile micro-architectural performance. We believe that these finer-grained measurements can provide more insights into resource requirement estimation for both training and inference, which are composed of these basic operations.

As opposed to DNNs, there are few profiling results reported for on-device deployment of traditional machine learning algorithms, and usually as a result of comparing these algorithms to newly developed ones. For example, inference time and energy are profiled in Reference [99] for a newly proposed tree-based Bonsai method, local deep kernel learning, single hidden layer neural network, and gradient boosted decision tree on the Arduino Uno micro-controller board. However, memory footprint is not profiled. Some other works empirically analyze the complexity and performance of machine learning algorithms [114, 207] where the experiments are conducted on computers but not resource constrained devices. To better understand the resource requirements of traditional machine learning methods, more systematic experiments need to be designed to profile their performance on different platforms and various frameworks.

3.1.3 Resource Modeling and Estimation. To provide more insights into how efficiently machine learning algorithms can be run on a given edge device, it is helpful to model and predict the re-

Table 4. DNN Profiling Benchmarks

Benchmark	Platform	Framework	Model	Metric	Highlight
AI Android [87]	57 SoC Processors; 200 mobile phones	Tensorflow Lite	CNNs for 9 image processing tests	inference: runtime	the most comprehensive benchmark on CNN deployment on Android system
NAS [24]	Intel i5-7600, NVidia Jetson TX1, Xiaomi Redmi Note 4	N/A	8 light-weight CNNs from neural architecture search methods	inference: time and memory	survey of models generated from NAS methods on resource-limited devices
Fathom [1]	Skylake i7-6700k CPU with 32GB RAM; NVidia GeForce GtX 960	TensorFlow	3CNNs, 2RNNs, 1DRL, 1Autoencoder, 1Memory Network	training: single-/multi-thread execution time by op type (e.g., MatMul, Conv2D, etc., 22 in total)	micro-architectural analysis; exploration of parallelism
BigDataBench [53]	Intel ®Xeon E5-2620 V3 CPU with 64GB RAM	Hadoop, Spark, JStorm, MPI, Impala, Hive, TensorFlow, Caffe (each supporting different models)	Micro benchmarks: 21 ops, e.g., sort, conv, etc.; Comp benchmarks:23, e.g., pagerank, kmeans, etc.	training: runtime, bandwidth, memory (L1-L3 cache and DRAM)	operating system level benchmarking; large variety of models
DAWNBench [28]	Nvidia K80 GPU	TensorFlow, PyTorch	ResNets	training: time-to-accuracy vs. optimization techniques, inference runtime vs. training runtime	defining a new metric by end-to-end training time to a certain validation accuracy
DeepBench [147]	Training:Intel Xeon Phi 7250, 7 types of NVidia GPUs; Inference: 3 NVidia GPUs, iPhone6 & 7, Raspberry Pi3	N/A	Operations: Matmul, Conv, Recurrent Ops, All-reduce	training and inference runtime, FLOPs	profiling basic operations with respect to input dimensions
TBD [215]	16 machines each with a Xeon 28-core CPU and 1-4 NVidia Quadro P4000 / NVidia TITAN Xp GPUs	TensorFlow, MXNet, CNTK	3CNNs, 3RNNs, 1GAN, 1DRL	training: throughput, GPU compute utilization, FP32 utilization, CPU utilization, memory	profiling of same model across frameworks; multi-GPU multi-machine training
SyNERGY [149]	NVidia Jetson TX1	Caffe	11 CNNs	inference: energy consumption (both per layer and network),SIMD instructions, bus access	proposing a multi-variable regression model to predict energy consumption
DNNAnalysis [17]	NVidia Jetson TX1	Torch7	14 CNNs	inference: runtime, power, memory, accuracy vs. throughput, parameter utilization	comparison of CNN inference on an embedded computing system
DNNBenchmarks [10]	NVidia Jetson TX1; NVidia TITAN Xp GPU	PyTorch	44 CNNs	inference:accuracy, memory, FLOPs, runtime	very comprehensive comparison of CNN inference on both embedded computing system and powerful workstation

Table 5. DNN Resource Requirements Modeling

Work	Platform	Framework	Metric	Measured features	Regression model	Relative error
Augur [120]	NVidia TK1, TX1	Caffe	inference: memory, time	matrix dimensions in matmul, weights, activations	linear	memory: 28% - 50%; time: 6% - 20%
Paleo [142]	NVidia Titan X GPU cluster	TensorFlow	training & inference: time	forward & backward FLOPs, weights, activations, data, platform percent of peak	linear	4%–30%
Gianniti et al. [57]	NVidia Quadro M6000 GPU	-	training: time	forward & backward FLOPs of all types of layers	linear	<23%
SyNERGY [149]	Nvidia Jetson TX1	Caffe	inference: energy	MACs	linear	<17% (w/o MobileNet)
NeuralPower [13]	Nvidia Titan X & GTX 1070	TensorFlow & Caffe	inference: time, power, energy	layer configuration hyper-parameters, memory access, FLOPs, activations, batch size	polynomial	time: <24%; power: <20%; energy: <5%
HyperPower [161]	Nvidia GTX1070 & Tegra TX1	Caffe	inference: power, memory	layer configuration hyper-parameters	linear	RMSPE < 7%
Yang et al. [203]	ASIC Eyeriss [23]	—	inference: energy	MACs, memory access	—	—
DeLight [150]	Nvidia Tegra TK1	Theano	training& inference: energy	layer configuration hyper-parameters	linear	—

ASIC: Application-Specific Integrated Circuit. Matmul: matrix multiplication. RMSPE: root mean square percentage error.

source requirements of these algorithms. Even though performance profiling can provide accurate evaluation of resource requirements, it can be costly as it requires the deployment of all models to be profiled. If such requirements can be modeled before deployment, then it will be more helpful for algorithm design and selection. Recently, there have been an increasing number of studies that attempt to estimate energy, power, memory and runtime of DNNs based on measures such as matrix dimension and MACs. There are many benefits if one can predict resource requirements without deployment and runtime profiling. First, it can provide an estimate of training and deployment cost of a model before actual deployment, which can reduce unnecessary implementation costs. Second, the modeled resource requirements can be integrated into the algorithm design process, which helps to efficiently create models that are tailored to specific user-defined resource constraints. Third, knowing the resource consumption of different operation types can help to decide when offloading and performance optimization are needed to successfully run learning models on edge devices [120].

In Table 5, we summarize recent studies that propose approaches to model resource requirements of DNNs. Generally, a linear regression model is built upon common features such as FLOPs for certain types of operations, activation size, matrix size, kernel size, layer configuration hyper-parameters, and so on, which are not hard to aquire. The Relative Error column shows how these estimation models perform compared to the actual runtime profiling results of a network. We can see that most models demonstrate a relative error between 20% and 30%, which indicates that even though hardware dependency and various measures are taken into account, it can still be challenging to make consistently accurate predictions on resource requirements. This large relative error further shows that metrics such as FLOPs or MACs alone are very crude estimates that can provide limited insights. Note that all models are platform and framework dependent, meaning that the coefficients of the regression model will change when the platform or framework changes. Nevertheless, these studies provide a feasible approach to approximate resource requirements of

DNNs. It remains to be seen whether similar methodology can be adopted to model the resource requirements of machine learning algorithms other than DNNs.

3.1.4 New Metrics. Resource constrained machine learning models cannot be evaluated solely on traditional performance metrics like, accuracy for classification, perplexity in language models, or mean average precision for object detection. In addition to these traditional metrics there is a need for system specific performance metrics. Such metrics can be broadly categorized as below:

(1) Analyze the multi-objective pareto front of the designed model along the dimension of accuracy and resource footprint. Following Reference [24] some commonly used metrics include,

$$\text{Reward} = \begin{cases} \text{Accuracy}, & \text{if Power} \leq \text{Power budget} \\ 0, & \text{otherwise}, \end{cases} \tag{1}$$

or

$$\text{Reward} = \begin{cases} 1 - \text{Energy}^*, & \text{if Accuracy} \geq \text{Accuracy threshold} \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where Energy* is a normalized energy consumption. Using this measure, a model can be guaranteed to fulfill the resource constraints. However, it is hard to compare several models as there is no single metric. In addition, it is also harder to optimize the model when reward goes to zero.

(2) Scalarization of the multi-objective metric into a unified reward [71], **Figure of Merit (FoM)** [183], or NetScore [198] that merges all performance measures into one number, e.g.,

$$\text{Reward} = -\text{Error} \times \log(\text{FLOPs}), \tag{3}$$

or

$$\text{FoM} = \frac{\text{Accuracy}}{\text{Runtime} \times \text{Power}}, \tag{4}$$

or

$$\text{NetScore} = 20 \log \left(\frac{\text{Accuracy}^{\alpha}}{\text{Parameters}^{\beta} \text{MACs}^{\gamma}} \right), \tag{5}$$

where α, β, and γ are coefficients that control the influence of individual metrics. Using a single value, a relative ordering between several models is possible. However, now there are no separate threshold on the resource requirements. Hence, the optimized model may still not fit the hardware limitations.

Note that, in addition to using the metrics in Equations (1) to (5) to evaluate the model performance, References [24, 71, 183] also utilize these metrics to build and optimize their learning algorithms. The main idea is to accurately maintain the system requirements and also improve the model's performance during training [171].

3.2 Resource Efficient Training

In this section, we review existing algorithm improvements on resource-constrained training. Note that orthogonal approaches have been made in specialized hardware architecture design and dataflow/memory optimization to enable on-device learning [148, 167]. While algorithmic and architectural approaches are tightly correlated and in most cases complement each other, we maintain our focus on resource-efficient algorithms that can work across platforms.

In the machine learning task, the available resources are mainly consumed by three sources: (1) the ML model itself, (2) the optimization process that learns the model, and (3) the dataset used for learning. For instance, a complex DNN has a large memory footprint while a kernel-based method may require high computational power. The dataset requires storage when it is

collected and consumes memory when it is used for training. Correspondingly, there are mainly three approaches that adapt existing ML algorithms to a resource-constrained setting: (1) reducing model complexity by incorporating resource constraints into the hypothesis class, (2) modifying the optimization routine to reduce resource requirements during training, and (3) developing new data representations that utilize less storage. Particularly, for data representation, there is a line of work that proposes new learning protocols for resource-constrained scenarios where full data observability is not possible [8, 157]. In the following, we broadly review resource efficient training methods according to these categories.

3.2.1 Lightweight ML Algorithms. As shown in Table 2 in Section 3.1.1, some traditional ML algorithms, such as Naive Bayes, SVMs, Linear Regression, and certain Decision Tree algorithms like C4.5, have relatively low resource footprints [2]. The lightweight nature of these algorithms make them good candidates for on-device learning applications. These algorithms can be readily implemented on a particular embedded sensor system or portable device with few refinements to fit the specific hardware architecture [67, 106, 118].

3.2.2 Reducing Model Complexity. Adding constraints or discovering structures is a popular approach to reducing the capacity of a hypotheses class. Although most such approaches target reduced footprint during inference, they may still yield advantages during training or subsequent re-trainings. Of course, there is an underlying assumption. As previously discussed in Section 3.1, an exact characterization of the resource footprints (memory or compute) in terms of the model hypotheses class is a non-trivial problem. Most algorithmic research simplify this problem by assuming an underlying Turing model,[1] where the computation or model complexity is represented in terms of $\sim O(|\mathcal{H}|)$, where, $O(\cdot)$ = Big O notation and $|\mathcal{H}|$ = a capacity measure for the hypothesis class like, sample sparsity for kernel methods, no. of model parameters for DNN, and so on. Under this assumption, reducing the model parameters or the hypotheses class' complexity is likely to yield lower resource footprint in terms of $O(\cdot)$ during training. Although, recent research has shown that in practicality simply reducing these model parameters do not always lead to efficient on-device learning [14, 204]. However, for the sake of completeness, we include these approaches that adopt reducing model complexity for improved resource footprints.

Decision trees can have large memory footprint due to the large number of tree nodes. To avoid over-fitting and to reduce this memory footprint, pruning is a typical approach applied for deploying decision trees [98]. Recently, there are also studies that develop shallow and sparse tree learners with powerful nodes that only require a few kilobytes of memory [99].

For ensemble-based algorithms such as boosting and random forest, a critical question is how to select weak learners to achieve higher accuracy with lower computational cost. To address this issue, Reference[62] proposed a greedy approach that selects weak decision-tree-based learners that can generate a prediction at any time, where the accuracy can increase if larger latency is allowed to process weaker learners.

For DNNs, there have been a plethora of studies that propose more lightweight model architectures for on-device learning. In Table 6, we provide a comparative overview of some representative lightweight CNN architectures. Compared to networks presented in Table 3, these networks are much smaller in terms of size and computation, while still retaining fairly good accuracy compared to large CNNs. Unfortunately, there is not much work developing lightweight models for DNN architectures other than CNNs.

Among the models presented in Table 6, MnasNet is a representative model generated via automatic **neural architecture search (NAS)**, whereas all other networks are designed manually. NAS

[1]See Sections 4.2.1 and 3.1.1 for more details.

Table 6. Comparison of Lightweight CNNs

Metric	MobileNet V1-1.0[80]	MobileNet V2-1.0[153]	Squeeze-Net[86]	Squeeze-Next-1.0-23[56]	ShuffleNet $1 \times g = 8$[212]	Condense-Net[84]	MnasNet [171]
Top-1 acc.	70.9	71.8	57.5	59.0	67.6	71.0	74.0
Top-5 acc.	89.9	91.0	80.3	82.3	-	90.0	91.8
Input size	224×224	224×224	224×224	227×227	224×224	224×224	224×224
# of stacked CONV layers	27	20	26	22	17	37	18
Weights	3.24M	2.17M	1.25M	0.62M	3.9M	2.8M	3.9M
Activations	5.2M	1.46M	4.8M	4.7M	3.2M	1.1M	3.9M
MACs	568M	299M	388M	282M	138M	274M	317M
# of FC layers	1	1	0	1	1	1	1
Weights	1M	1.3M	0	0.1M	1.5M	0.1M	0.3M
Activations	2K	2.3K	0	1.1K	2.5K	1.1K	1.3K
MACs	1M	1.3M	0	0.1M	1.5M	0.1M	0.3M
Total weights	4.24M	3.47M	1.25M	0.72M	5.4M	2.9M	4.2M
Total activations	5.2M	1.46M	4.8M	4.7M	3.2M	1.1M	3.9M
Total MACs	569M	300M	388M	282M	140M	274M	317M

is usually based on reinforcement learning or evolutionary algorithms where a predefined search space is explored to generate the optimal network architecture that achieves the best tradeoff between accuracy and energy/runtime [196]. To speed up NAS, the resource modeling techniques mentioned in Section 3.1.3 could be helpful, as they can replace the computationally intensive process of measuring the actual model performance on device. For example, in ChamNet [32], predictors for model accuracy, latency, and energy consumption are proposed that can guide the search and reduce time to find desired models from many GPU hours to mere minutes.

Note that even though for most NAS algorithms, the goal is to optimize the inference performance; the resulting architectures usually contain fewer parameters and require fewer FLOPS. This follows from the above-mentioned assumption of the equivalence between model parameters and (asymptotic) resource footprints. In fact, a recent study [143] shows that training time can be significantly reduced on the architectures found in their proposed search space. If needed, on-device resource constraints that are specific to model training can be taken into account by NAS. Keeping this in consideration, we include NAS in the resource-efficient training section, as they provide good candidates for on-device model training tasks.

Figure 3 shows a chronological plot of the evolution of CNN architectures. We include the models in Table 3, Table 6, as well as recent NAS works including FBNet [199], FBNetV2 [185], ChamNet [32], ProxylessNAS [15], SinglePath-NAS [162], EfficientNet [172], MixNet [173], MobileNetV3 [79], and RegNet [143]. This plot shows the FLOPS each model requires, and only models that achieve more than 60% top-1 accuracy on the ImageNet dataset are presented. Figure 3 shows a clear transition from hand-designed CNN architectures to NAS over time, with the best performing models all being generated using NAS techniques. An advantage of NAS over hand-designed architectures is that different accuracy-resource tradeoffs can be made a part of the search space resulting in a group of models that are suitable for platforms with various resource constraints.

Apart from designing new architectures from scratch, another approach to reducing DNN model complexity is to exploit the sparse structure of the model architecture. One efficient way to learn sparse DNN structures is to add a group lasso regularization term in the loss function that encourages sparse structures in various DNN components like filters, channels, filter shapes, layer depth, and so on [49, 105, 193]. Particularly in Reference [193], when designing the regularization term, the authors take into account how computation and memory access involved in matrix

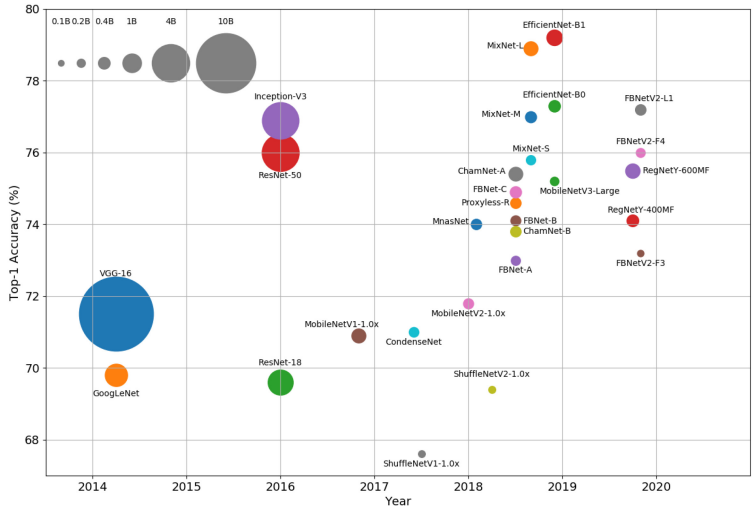

Fig. 3. Ball chart of the chronological evolution of model complexity. Top-1 accuracy is measured on the ImageNet dataset. The model complexity is represented by FLOPS and reflected by the ball size. The accuracy and FLOPS are taken from original publications of the models. The time of the model is when the associated publication is first made available online.

multiplication are executed on hardware platforms such that the resulting DNN network can achieve practical computation acceleration.

Another approach adopted for reduced model complexity involves quantization by using low or mixed precision data representation. Quantization was first applied to inference where only the weights and/or activations are quantized post-training. This approach was then extended to training where gradients are also quantized. Conventionally, all numerical values including weights, data, and intermediate results are represented and stored using 32-bit floating point data format. There are a plethora of emerging studies exploring the use of 16-bit or lower precision for storing some or all of these numerical values in model training without much degradation in model accuracy [65, 129]. Most of these approaches involve modifications in the training routine to account for the quantization error. A more detailed discussion on the training routine modifications of these methods is provided in the next Section.

3.2.3 Modifying Optimization Routines. There are broadly two directions of research aimed at improving the performance of quantized models.

Resource constrained Model-Centric Optimization Routines: Optimization or other numerical computation involved in training certain traditional ML models can be very resource intensive. For example, for kernel-based algorithms such as SVM, kernel computation usually consumes the most energy and memory. Efficient algorithms have been proposed to reduce the computation complexity and memory footprint of kernel computations and are beneficial for both training and inference [81, 92, 103].

The choice of optimization strategy is crucial for DNN training, as a proper optimization technique is critical to finding a good local minimum. While many efficient optimization algorithms such as Adam as well as network initialization techniques are exploited in DNN training, the main objective is to improve convergence and consequently reduce training time. Other resource constraints are not taken into account explicitly. In this part, we focus on efficient training algorithms

that consider performance other than runtime and could potentially enable DNN training on devices with limited memory and power.

The most popular approach for reducing memory footprint of DNN training is quantization, as mentioned in Section 3.2.2. However, quantization inevitably introduces error, which could make training unstable. To resolve this issue, recent works propose minor modifications to the model training process. A popular approach involves designing a mathematical model to characterize the statistical error introduced when limiting the model's precision through rounding or quantization [26, 78, 93, 201]; and proposing improvements [77, 96, 122]. In fact, most of the recent research in this line involves modifications in training updates either through stochastic rounding, weight initialization or through introducing the quantization error into gradient updates. The work in Reference [66] uses fixed point representation and stochastic rounding to account for quantization error. Reference [29] further limits the precision to binary representation of weights using stochastic binarization during forward and back propagation. However the full precision is maintained during gradient updates. The authors claim that such unbiased binarization lends to additional regularization of the network. A more detailed theoretical analysis of stochastic rounding and binary connect networks can be found in Reference [110]. The work in Reference [117] introduced a quantized version of back-propagation by representing the weights as powers of 2. This representation converts the multiplication operations to cheaper bit-shift operations. An alternative noisy back propagation algorithm to account for the error due to binarization of weights is also introduced in Reference [95]. In a slightly different approach [85] uses a deterministic quantization but introduces a "straight through estimator" for gradient computation during back propagation. They further introduce a shift-based batch normalization and a new shift-based AdaMax algorithm. Reference [144] introduces a new mechanism to compute binarized convolutions and, additionally, introduces a scaling for the binarized gradients. Reference [195] applies a non-subtractive dither quantization function to gradients and shows that this function can induce sparsity and non-zero values with low bitwidth for large-enough quantization stepsize.

A more recent approach in Reference [187] moves away from fixed-points to a new floating point representation with stochastic rounding and chunk-based accumulation during training. Finally, recent work in Reference [44, 90] simulates the effect of quantization during inference and adds correction to the training updates by introducing quantization noise in the gradient updates. Slightly different from Reference [90] where all the weights are quantized, Reference [44] quantizes a random subset of weights and back propagates unbiased gradients for the complimentary subset.

Apart from the theoretical development of methods that reduce quantization error, some other works focus on making quantization more practical and generalizable. Reference [213] generalizes the method of binarized neural networks to allow arbitrary bit-widths for weights, activations, and gradients, and stochastically quantizes the gradients during the backward pass. To generalize the quantization technique to different types of models, Reference [16] introduces two learnable statistics of DNN tensors, namely, shifted and squeezed factors, which are used to adjust the tensors' range in 8-bits for minimizing quantization loss. They show that their method works out-of-the-box for large-scale DNNs without much tuning. However, in most works, there are still some computation in training that requires full precision representation. To address this issue, Reference [206] proposes a framework that quantizes the complete training path including weights, activations, gradients, error, updates, and batch-norm layers, and converts them to 8-bit integers. Different quantization functions are used for different compute elements.

In a very different approach, Reference [159] proposes **expectation back propagation (EBP)**, an algorithm for learning the weights of a binary network using a Variational Bayes technique. The algorithm can be used to train the network such that, each weight can be restricted to be binary or

ternary values. This approach is very different from the above gradient-based back-propagation algorithms. However, this approach assumes that the bias is real and is not currently applicable to CNNs. A chronological summary of the approaches modifying the training algorithms to account for quantization error is provided in Table 7. While the above approaches (summarized in Table 7) mainly target modifications to account for the limited bit representations (due to quantization), in the following section, we highlight several other approaches that are not limited to quantization corrections.

Apart from quantization, there are mainly two other approaches targeted at reducing memory footprint of DNN training, namely, layerwise training [20, 61], and trading computation for memory [63]. One major cause for the heavy memory footprint of DNN training is end-to-end back-propagation, which requires storing all intermediate feature maps for gradient calculation. Sequential layer-by-layer training has been proposed as an alternative [20]. While this method was originally proposed for better DNN interpretation, it requires less memory usage while retaining the generalization ability of the network. Trading computation for memory reduces memory footprint by releasing some of the memory allocated to intermediate results, and recomputing these results as needed. This approach is shown to tightly fit within almost any user-set memory budget while minimizing the computational cost [63]. Apart from these two approaches, some other studies focus on exploiting the specific DNN architectures and targeted datasets, and propose methods using novel loss functions [22, 208], and training pipelines [22] to improve the memory and computation efficiency of DNNs.

While the aforementioned studies demonstrate promising experimental results, they barely provide any analysis on the generalization error nor the sample complexity under a limited memory budget. A few studies target at this issue by developing memory-bounded learning techniques with some performance guarantees. For example, References [58, 102, 163, 170] proposed memory-bounded optimization routines for sparse linear regression and provided upper bounds on regret in an online learning setting. For PCA, [4, 109, 131, 205, 211] proposed memory or computation-efficient optimization algorithms while guaranteeing optimal sample complexity.

Note that in an environment with a cluster of CPUs or GPUs, the most popular technique for speeding up training is parallelization, where multiple computational entities share the computation and storage for both data and model [39]. However, in IoT applications with embedded systems, it is not common to have similar settings as a computer cluster. Usually the devices are scattered and operated in a distributed fashion. To enable cooperation among a group of devices, distributed learning techniques such as consensus, federated learning and many others have been proposed [127, 176]. However, the primary concern of these techniques are data privacy and the lack of central control entity in the application field, which is out of the scope of this article.

Resource constrained Generic Optimization Routines: While the methods in the previous subsection are designed for specific algorithms, the approaches in this section target generic optimization routines. One of the first works in this line of research is BuckWild! [38], which introduces low precision SGD and provides theoretical convergence proofs of the algorithm. An implementation of the Buckwild! algorithm through a computation model called "Dataset Model Gradients Communicate" is provided in Reference [36]. As an improvement to the Buckwild! approach, the work in Reference [37] proposes novel "bit-centering" quantization for low precision SGD and **stochastic variance reduction gradient (SVRG)**. An alternative approach to improve upon low-precision SGD in Reference [38] is also introduced in Reference [202]. This approach introduces the SWALP algorithm, an extension of the stochastic weight averaging scheme for SGDs for low precision arithmetic. While most of the above approaches mainly analyze the effect of quantization to SGD and its variants; Reference [165] adopts an approach of improving the SGD algorithm under sparse representation. Finally, in a more recent work, Reference [113]

Table 7. The Chronology of the Recent Approaches that Modifies the Training Algorithm to Account for Quantization Error

Year	Approach	Keywords	Quantization[1]			Benchmark	
			Forward	Backward	Parameter Update	Data	Model
2014	EBP [159]	Expectation Back Propagation	1 bit, FP	—	—	used in [31]	Proprietary MLP
2015	Gupta et al. [66]	Stochastic Rounding	16 bits	16 bits	16 bits	MNIST	Proprietary MLP , LeNet-5
			20 bits	20 bits	20 bits	CIFAR-10	used in [76]
	Binary Connect [29]	Stochastic Binarization	1 bit	1 bit	Float 32[2]	MNIST CIFAR-10 SVHN	Proprietary MLP, CNN
2016	Lin et al. [117]	Stochastic Binarization No forward pass multiplication Quantized back propagation	1 bit	1 bit	Float 32	MNIST CIFAR-10 SVHN	Proprietary MLP, CNN
	Bitwise Net [95]	Weight Compression Noisy back propagation	1 bit	1 bit	1 bit Float 32[3]	MNIST	Proprietary MLP
	XNOR-Net [144]	Binary convolution Binary dot-product Scaling binary gradient	1 bit	1 bit	1 bit Float 32[4]	ImageNet	AlexNet ResNet-18 GoogLenet
	DoReFa-Net [213]	stochastic gradient quantization arbitrary bit-width	1-8 bit	1-8 bit	2-32 bit	SVHN	proprietary CNN
						ImageNet	AlexNet
2017	QNN [85]	Deterministic binarization Straight through estimators to avoid saturation Shift-based Batch Normalization Shift-based AdaMAX	1 bit	1 bit	1 bit[5]	MNIST	proprietary MLP
						CiFAR-10 SVHN	CNN from [29]
						ImageNet	AlexNet GoogLenet
			4 bit	4 bit	4 bit[6]	Penn Treebank	proprietary RNN LSTM
2018	Wang et al. [187]	novel floating point chunk-based accumulation stochastic rounding	8 bit	8 bit	8 bit[7]	CIFAR-10	proprietary CNN ResNET
						BN50 [178]	proprietary MLP
						ImageNet	AlexNet ResNET18 ResNET50
	Jacob et al. [90]	training with simulated quantization	8 bit	8 bit	8 bit[8]	Imagenet	Resnet Inception v3 MobileNet
						COCO	MobileNet SSD
						Flickr [80]	MobileNet SSD
2019	WAGEUBN [206]	batch-norm layer quantization 8-bit integer representation combination of direct, constant and shift quantization	8 bit	8 bit	8 bit	ImageNet	ResNet18/34/50
2020	S2FP8 [16]	shifted and squeezed FP8 representation of tensors tensor distribution learning	8 bit	8 bit	32 bit	CIFAR-10	ResNet20/34/50
						ImageNet	ResNet18/50
						English-Vietnamese	Transformer-Tiny
						MovieLens	Neural Collaborative Filtering (NCF)
	Wiedemann et al. [195]	stochastic gradient quantization induce sparsity non-subtractive dither	8 bit	8 bit	32 bit	MNIST	LeNet
						CIFAR-10/100	AlexNet ResNet18 VGG11
						ImageNet	ResNet18
	Quant-Noise [44]	training using quantization noise	8 bit	8 bit	8 bit	Wikitext-103 MNLI ImageNet	RoBERT RoBERT EfficientNet-B3

[1] Minimum quantization for best performing model reported.
[2] All real valued vectors are reported as Float 32 by default.
[3] Involves tuning a separate set of parameters with floating point precision.
[4] Becomes Float 32 if gradient scaling is used.
[5] Except the first layer input of 8 bits.
[6] Contains results with 2 bit, 3 bit and floating point precision.
[7] Additional 16 bit for accumulation.
[8] Uses 7 bit precision for some Inception v3 experiments.

provides the first dimension-free bound for the convergence of low precision SGD algorithms. There has been significant research on analyzing traditional first order algorithms like SGD, SVRG, and so on, in parallel and distributed settings with low precision/quantized bit representation [3, 52, 116, 125, 166, 174, 192, 194, 200]. Even though these approaches generalize to single device learning, the main focus of these research is in reducing the communication bottleneck for parallel and distributed settings. This area is not within the scope of the current survey. A more detailed survey on communication-efficient distributed optimization is available in References [64, 111, 115, 175].

An alternative line of work involves implementing fixed point **Quadratic Programs (QP)** for solving linear **Model Predictive Control (MPC)**. Most of these algorithms involve modifying the fast gradient methods developed in Reference [140] to obtain a suboptimal solution in a finite number of iterations under resource constrained settings. In fact these algorithms extend the generic Interior point methods, Active-set methods, Fast Gradient Methods, Alternating Direction method of multipliers and Alternating minimization algorithm, to handle quantization error introduced due to fixed point implementations. However, all these approaches are targeted toward linear MPC problems, and is not the main focus of this survey. A comprehensive survey on these fixed point QPs for linear MPCs is available in Reference [126].

3.2.4 Data Compression. Besides complexity in models or optimization routines, the dimensionality and volume of training data significantly dictates the algorithm design for on-device learning. One critical aspect is the *sample complexity*,[2] i.e., amount of training data needed by the algorithm to achieve a desired statistical accuracy. This aspect has been widely studied by the machine learning community and has led to newer learning settings like Semi-Supervised learning [19], Transductive learning [19, 180], Universum learning [42, 180], Adversarial learning [60], Learning under Privileged Information [180], Learning Using Statistical Invariants [181] and many more. These settings allow for the design of advanced algorithms that can achieve high test accuracies even with limited training data. A detailed coverage of these approaches is outside the scope of this survey but readers are directed to References [19, 25, 42, 55, 60, 179–181, 216] for a more exhaustive reading.

In this article, we focus on those approaches that reduce the resource footprint imposed by the dimensionality and the volume of training data.

For example, the storage and memory requirement of kNN approaches are large due the fact that all training samples need to be stored for inference. Therefore, techniques that compress training data are usually used to reduce the memory footprint of the algorithm [100, 188, 210]. Data compression or sparsification is also used to improve DNN training efficiency. Some approaches exploit matrix sparsity to reduce the memory footprint to store training data. For example, Reference[151] transforms the input data to a lower-dimensional embedding that can be further factorized into the product of a dictionary matrix and a block-sparse coefficient matrix. Training using the transformed data is shown to be more efficient in memory, runtime and energy consumption. In Reference [112], a novel data embedding technique is used in RNN training, which can represent the vocabulary for language modeling in a more compressed manner.

3.2.5 New Protocols for Data Observation. For most learning algorithms, a major assumption is that full i.i.d. data are accessible in batch or streaming fashion. However, under certain constraints, only partial features of part or the entire dataset can be observed. For example, feature extraction can be costly due to either expensive feature computation or labor-intensive feature acquisition. Another instance is that memory is not sufficient to store a small batch of high-dimensional data

[2]Formal introduction provided in Section 4.2.1.

samples. This problem leads to studies on limited attribute observation [8, 18], where both memory and computation are limited by restricted observation of sample attributes. Under this setting, a few studies propose new learning algorithms and analyze sample complexity for applications such as sparse PCA [157], sparse linear regression [18, 69, 89, 137], parity learning [164], and so on. While this new learning protocol provides valuable insights into the tradeoffs of sample complexity, data observability, and memory footprint, it remains to be explored how they can be used to design resource-efficient algorithms for a wider range of ML models.

3.3 Resource Efficient Inference

While inference is not the main focus of this article, for the sake of completeness, we provide a brief overview of resource efficient approaches that enable on-device inference. Due to the fact that most traditional ML algorithms are not resource intensive during inference, we will only focus on methods proposed for DNNs.

Apart from designing lightweight DNN architectures as discussed in Section 3.2.2, another popular approach is static model compression, where, during or after training, the model is compressed in size via techniques such as network pruning [68, 119], vector quantization [59, 68, 186], distillation [75], hashing [21], network projection [145], binarization [30], and so on. These studies demonstrate significant reduction in model size while retaining most of the network's predictive capacity.

However, since these models will not change after compression, their complexities cannot further adapt to dynamic on-device resources or inputs. To address this issue, some novel adaptive techniques for faster inference on embedded devices are explored in recent works. For example, Reference [41] uses dynamic layerwise partitioning and partial execution of CNN-based model inference, allowing for more robust support of dynamic sensing rates and large data volume. Another approach proposed in Reference [123] involves developing a low cost predictive model that dynamically selects models from a set of pre-trained DNNs by weighing desired accuracy and inference time as metrics for embedded devices. Reference [107] introduced Neuro.ZERO to provide energy and intermittence aware DNN inference and training along with adaptive high-precision fixed-point arithmetic to allow for accelerated run-time embedded hardware performance.

There have been observations that not all inputs require the same amount of computational power to be processed. A simple model may be sufficient to classify samples that are easy to distinguish, while a more complex model is only needed to process difficult samples. Based on this reasoning, References [11, 83, 190] focus on building and running adaptive models that dynamically scale computation according to inputs. The specific methods proposed in these works include early termination, exploration of cascaded models, and selectively using/skipping parts of the network. These adaptive approaches are complementary to the compression approach, and further resource efficiency can be expected if these two approaches are applied altogether.

Apart from the "model-centric" approaches as discussed in Section 3.2.3, hardware and system optimizations have also been exploited for efficient model deployment. To explore how the choice of computing devices can impact performance of these models on the edge we direct the reader to Reference [138]. The survey discusses how various low power hardware such as ASICs, FPGAs, RISC-V, and embedded devices are used for efficient inferencing of Deep Learning models. Another interesting survey [189] expands on the various communication and computation modes for deep learning models where edge devices and the cloud server work in tandem. Their work explores concepts such as integral offloading, partial offloading, vertical collaboration, and horizontal collaboration to allow efficient edge-based inference in collaboration with the cloud. Reference [214] further advances these ideas from the point of view of the network latency between the cloud server and edge device, exploring techniques such as model partitioning, edge caching,

input filtering, early exit strategies, and so on, to provide efficient on device inference. Lately, the embedded ML research community has also focused on interconnected and smart home devices for processing data privately and locally with stronger privacy restrictions and lower latencies. Reference [82] aggregates processing capability of potential embedded devices at home with comparisons between processing on mobile phones and specific hardware for efficient DL processing such as Coral TPU and the NVIDIA Jetson Nano.

3.4 Challenges in Resource-efficient Algorithm Development

From the discussions in this section, we observe the following challenges for resource characterization of existing ML algorithms and development of new resource-efficient algorithms:

(1) **Hardware dependency:** As observed from existing hardware platform benchmarks and resource requirement modeling approaches, resource characterization greatly depends on the platform, framework, and the computing library used. For example, on the Nvidia Jetson TX1 platform, the memory footprint of a CNN model can be very different based on the frameworks (Torch vs. Caffe) used [17, 149]. Among all resource constraints, memory footprint is particularly hard to quantify as many frameworks and libraries deploy unique memory allocation and optimization techniques. For example, MXNet provides multiple memory optimization mechanisms and the memory footprint is very different under each mechanism [139]. Consequently, given specific on-device resource budgets and performance requirements, it is challenging to choose the optimal algorithm that fits in the resource constraints. With all the variability and heterogeneity in implementation, it is thereby important to draw insights and discover trends on what is invariant across platforms. As discussed in Reference [124], cross-platform models that account for chip variability are needed to transfer knowledge from one type of hardware to another. Furthermore, we postulate that cross-framework or cross-library models are also important to enable resource prediction for all types of implementations.

(2) **Metric design:** While a few novel metrics are proposed for on-device ML algorithm analysis, they either cannot be used directly for algorithm comparison, or cannot provide guarantees that the training process will fit the resource budget. Particularly, for metrics devised based on hardware agnostic measures such as number of parameters or FLOPs, they rarely accurately reflect realistic algorithm performance on a specific platform. Therefore, more practical, commensurable and interpretable metrics need to be designed to determine which algorithm can provide the best tradeoff between accuracy and multiple resource constraints.

(3) **Algorithm focus:** Most studies on on-device learning focus on DNNs, especially CNNs, because their layerwise architectures carry a declarative specification of computational requirements. In contrast, there has been very little focus on RNNs and traditional machine learning methods. This is mainly due to their complexity or heterogeneity in structure or optimization method, and lack of a benchmarking dataset. However, as observed in Table 4, the edge platforms used for DNN profiling are generally mobile phones or embedded computing cards that still have gigabytes of RAM and large computational power. For IoT or embedded devices with megabytes or even kilobytes of RAM and low computational power, DNNs can barely fit. Therefore, traditional machine learning methods are equally important for edge learning, and their resource requirements and performance on edge devices need to be better profiled, estimated and understood. In addition, designing advanced learning algorithms that require only small amounts of training data to achieve high test accuracies is a huge challenge. Addressing this area can significantly improve the resource footprint

Table 8. Broad Categorization of Resource Constrained Algorithms in Section 3 with Respect to Their Underlying Learning Theories

Theory	Algorithms
Traditional Machine Learning Theory	Low-Footprint ML algorithms (Section 3.2.1) Reducing model-complexity (Section 3.2.2) Modifying Optimization routines (Section 3.2.3) Data Compression (Section 3.2.4)
Novel resource constrained machine learning theory	New protocols for data observation (Section 3.2.5)

(i.e., low memory footprint to store the training data), while maintaining highly accurate models.

(4) **Dynamic resource budget:** Most studies assume that the resource available for a given algorithm or application is static. However, resource budget for a specific application can be dynamic on platforms such as mobile phone due to events such as starting or closing applications and application priority changes [45]. Contention can occur when resource is smaller than the model requirements and resource can be wasted when it is abundant. To solve this issue, Reference [45] proposed an approach for multi-DNN applications. When trained offline, for each application, five models with diverse accuracy-latency tradeoffs are generated and their resource requirements are profiled. The models are nested with shared weights to reduce memory footprint. During deployment, a greedy heuristic approach is used to choose the best models for multiple applications at runtime that achieve user-defined accuracy and latency requirements while not exceeding the total memory of the device.

4 THEORETICAL CONSIDERATIONS FOR ON-DEVICE LEARNING

While Section 3 categorizes the approaches used for building resource-constrained machine learning models, this section focuses on the computational learning theory that is used to design and analyze these algorithms. Nearly every approach in the previous section is based on one of the following underlying theories,

- **Traditional machine learning theory:** Most of the existing approaches used to build the resource efficient machine learning models for inference (in Section 3.3) follow what is canonically referred to as traditional machine learning theory. In addition, approaches like, *low-footprint machine learning* (Section 3.2.1), *reducing model complexity* (Section 3.2.2), *data compression* (Section 3.2.4), *modifying optimization routines* (Section 3.2.3) used for resource efficient training, also follow this traditional learning theory. For example, low-footprint machine learning involves using algorithms with inherently low resource footprints designed under traditional learning theory. Approaches like *reducing model complexity* and *data compression*, incorporates additional resource constraints for algorithms designed under traditional learning theory. Finally, the approach of *modifying optimization routines* simply modifies the optimization algorithms used to solve the machine learning problem designed under such theories.
- **Novel resource constrained machine learning theory:** These approaches highlight the gaps in traditional learning theory and propose newer notions of learnability with resource constraint settings. Most of the algorithms in *new protocols for data observation* (Section 3.2.5) fall in this category. Typically, such approaches modify the traditional assumption of i.i.d. data being presented in a batch or streaming fashion and introduces a specific protocol of data observability. Additional algorithmic details are available in Section 3.2.5.

In this section, we provide a detailed survey of the advancements in the above categories. Note that there is an abundance of literature addressing traditional learning theories [132, 155, 180]. For completeness, however, we still include a very brief summary of such traditional theories in Section 4.2.1. This summary also acclimatizes the readers with the notations used throughout this section.

Section 4.1 formalizes the learning problem for both traditional as well as resource constrained settings. Next, Section 4.2.1 provides a brief survey of some of the popular traditional learning theories. The advanced learning theories developed for resource-constraint settings are provided in Section 4.2.2. Finally, we conclude by discussing the existing gaps and challenges in Section 4.3.

4.1 Formalization of the Learning Problem

Before delving into the details of the learning theories, we first formalize the learning problem for traditional machine learning algorithms. We then extend this formalization to novel machine learning algorithms under resource constraint settings in Section 4.1.2. For simplicity, we focus on supervised problems under inductive settings [25, 180]. Extensions to other advanced learning settings like transductive, semi-supervised, universum, and so on, can be found in References [132, 155, 180].

4.1.1 Traditional Machine Learning Problem. The supervised learning problem assumes an underlying label generating process $y = g(\mathbf{x})$, where $y \in \mathcal{Y}$ and $\mathbf{x} \in \mathcal{X}$(data domain) $\subseteq \mathbb{R}^d$; which is characterized by a data generating distribution $\mathcal{D}$. Under the inductive learning setting, the machine learning problem can be formalized as

Definition 1. **Inductive Learning**

Given. independent and identically distributed (i.i.d.) training samples $S = (\mathbf{x}_i, y_i)_{i=1}^n$ from the underlying data generating distribution $S \sim \mathcal{D}^n$, and a predefined loss function $l : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{O}$ (output domain).

Task. estimate a function/hypothesis $\hat{h} : \mathcal{X} \rightarrow \mathcal{Y}$ from a set of hypotheses $\mathcal{H}$ (a.k.a Hypothesis class), that best approximates the underlying data generating process $y = g(\mathbf{x})$, i.e., which minimizes

$$\mathbb{E}_{\mathcal{D}}(l(y, \hat{h}(\mathbf{x})) = R(\hat{h}). \tag{6}$$

Here,

$\mathbb{E}_{\mathcal{D}}(\cdot)$ = Expectation operator under distribution $\mathcal{D}$; $\mathbb{1}_{(\cdot)} = Indicator function$

and $R(\cdot)$ = True risk of a hypothesis

Some popular examples following this problem setting are

- Binary Classification problems using 0/1 loss where $\mathcal{Y} = \{-1, +1\}, \mathcal{O} = \{0, 1\}$ and $l(y, \hat{h}(\mathbf{x})) = \mathbb{1}_{y \neq \hat{h}(\mathbf{x})}$.
- Regression problems with additive gaussian noise (i.e., least-square loss) where $\mathcal{Y} = \mathbb{R}$, $\mathcal{O} = \mathbb{R}$ and $l(y, \hat{h}(\mathbf{x})) = (y - \hat{h}(\mathbf{x}))^2$.

Typically users incorporate apriori information about the domain while constructing the hypothesis class. In the simplest sense, the hypothesis class gets defined through the methodologies used to solve the above problems. For example, solving the regression problem using

- *least-squares* linear regression: the hypothesis class includes all *linear* models, i.e., $\mathcal{H} = \{h : h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b; \mathbf{w} \in \mathbb{R}^d; b \in \mathbb{R}\}$.

- *lasso L1−* linear regression: the hypothesis class includes all *linear* models of the form: $\mathcal{H} = \{h : h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b; \mathbf{w} \in \mathbb{R}^d; b \in \mathbb{R}; ||\mathbf{w}||_1 \leq B\}$.

4.1.2 Resource Constrained Machine Learning Problem. In addition to the goals highlighted in the above section, building resource constrained machine learning algorithms introduces additional constraints to the learning problem. The goal now is not just to minimize the true risk but also ensure that the resource constraints discussed in Section 2 are met. Typically, these resource constraints are imposed during *inference* or *training* phase of a machine learning pipeline. A mathematical formalization of this problem can therefore be given as,

Definition 2. **Resource Constrained Machine Learning**

Given. i.i.d. training samples $S = (\mathbf{x}_i, y_i)_{i=1}^n$ from the underlying data generating distribution $S \sim \mathcal{D}^n$, a predefined loss function $l : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{O}$, and a predefined resource constraint $C(\cdot)$.

Inference. estimate a function $\hat{h} : \mathcal{X} \rightarrow \mathcal{Y}$ that best approximates the underlying data generating process, i.e., Equation (6) and simultaneously satisfies $C(\hat{h})$.

Training. estimate a function $\hat{h} : \mathcal{X} \rightarrow \mathcal{Y}$ that best approximates the underlying data generating process, i.e., Equation (6) and simultaneously satisfies $C(A(S))$. Here, $A : S \rightarrow \mathcal{H}$ is an algorithm (model building process) that inputs the training data and outputs a hypothesis $\hat{h} \in \mathcal{H}$.

As seen above the main difference of this setting with respect to the traditional inductive learning setting is the additional constraint $C(\cdot)$ on the final model (during inference) or the Algorithm (for training). As an example, consider the least squares linear regression example discussed above. Typical resource constraints for such a problem may look like

- Memory constraint during inference enforced by requiring the model parameters $(\mathbf{w}, b)$ of the estimated model $y = \mathbf{w}^T\mathbf{x} + b$ be represented using float32 precision ($\sim 2^{32(d+1)}$ bit memory footprint) or int16 precision ($\sim 2^{16(d+1)}$ bit memory footprint); with $\mathbf{x} \in \mathbb{R}^d$. This is equivalent to a constraint on the final model $C_{\text{float32}}(\hat{w}, \hat{b}) = \{\hat{w} \in \{-3.4E + 38, \ldots, 3.4E + 38\}^d; \hat{b} = \{-3.4E + 38, \ldots, 3.4E + 38\}\}$ or $C_{\text{int16}}(\hat{w}, \hat{b}) = \{\hat{w} = \{-32768, \ldots, 32767\}^d; \hat{b} = \{-32768, \ldots, 32767\}\}$.
- Computation constraint during inference can be enforced by requiring $\hat{w}$ to be $k-$ sparse with $k < d$. This, would ensure $k + 1$ FLOPS per sample and equivalently constraints final model as $C(\hat{w}, \hat{b}) = \{\hat{w} \in \mathbb{R}^k\}$.

Similar, memory/computation constraint can be imposed onto the learning algorithm $A(S)$ during training.

4.2 Learning Theories

In this section, we discuss the learning theories developed for the problems discussed above.

4.2.1 Traditional Learning Theories. These theories target the traditional learning problem discussed in Section 4.1. Most traditional learning theories provide probabilistic guarantees of the goodness of a model (actually guarantees for all models in the Hypothesis class) with respect to the metric in Equation (6). Such theories typically decompose Equation (6) into two main components,

$$R(h) - R^* = \underbrace{(\min_{h \in \mathcal{H}} R(h) - R^*)}_{\text{approximation error}} + \underbrace{(R(h) - \min_{h \in \mathcal{H}} R(h))}_{\text{estimation error}}. \tag{7}$$

Here,

$$\text{Bayes Error, } R^* = \min_{\text{all measurable } h} R(h)$$
$$\text{and } \min_{h \in \mathcal{H}} R(h)) = \text{smallest in-class error}$$

The approximation error in Equation (7), typically depends on the choice of the hypothesis class and the problem domain. The value of this error is problem dependent. Traditional learning theories then characterize the estimation error. One of the fundamental theoretical frameworks used for such analyses is the PAC-learning framework.

Definition 3. **(Efficient) Agnostic PAC-learning** [132]
A hypothesis class $\mathcal{H}$ is agnostic PAC-learnable using an algorithm $A(S)$, if for a given $\epsilon, \delta \in (0, 1)$ and for any distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ (with $\mathcal{X} \subseteq R^d$) there exists a polynomial function $p_1(\frac{1}{\epsilon}, \frac{1}{\delta}, d, |\mathcal{H}|)$ such that for any sample size $n \geq p_1(\frac{1}{\epsilon}, \frac{1}{\delta}, d, |\mathcal{H}|)$ the following holds:

$$\mathbb{P}_{\mathcal{D}}(R(h_S) - \min_{h \in \mathcal{H}} R(h) \leq \epsilon) \geq 1 - \delta; \quad \textit{where } h_S = A(S) \in \mathcal{H}. \tag{8}$$

Further, it is efficiently PAC learnable if the computation complexity is $O(p_2(\frac{1}{\epsilon}, \frac{1}{\delta}, d, |\mathcal{H}|))$ for some polynomial function p_2.

As seen from Definition 3 there are two aspects of PAC-learning.
Sample Complexity characterizing Generalization Bounds: Although Definition 3 provides the learnability framework for a wide range of machine learning problems like regression, multi-class classification, recommendation, and so on. For simplicity, in this section we mainly focus on the *sample-complexity* (and equivalently the generalization bounds) for binary classification problems. Extensions of these theories for other learning problems can be found in References [132, 155].

Definition 3 guarantees that an estimated model using the algorithm $A(S)$ and sample size $n \geq p_1(\frac{1}{\epsilon}, \frac{1}{\delta}, d, |\mathcal{H}|)$, will guarantee predictions with error tolerance of ϵ and a probability $1 - \delta$. One popular class of algorithm choice is the **Empirical Risk Minimization- (ERM)** based algorithms. Here, the algorithms return the hypothesis that minimizes an empirical estimate of the risk function in Equation (6) given by,

$$\text{ERM estimate } h_S = ERM(S) = \underset{h \in \mathcal{H}}{argmax}\ R_S(h), \tag{9}$$

$$\text{where Empirical Risk } R_S(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{y_i \neq h(\mathbf{x}_i)}.$$

A very interesting property of the ERM-based algorithms is the *uniform convergence* property (see Reference [155]). This property dictates that for the ERM algorithm to return a *good* model from within $\mathcal{H}$, we need to bound the term $|R_S(h) - R(h)|; \forall h \in \mathcal{H}$. In fact, most popular learning theories characterize the number of samples (a.k.a sample complexity) needed for bounding this term $|R_S(h) - R(h)|; \forall h \in \mathcal{H}$ for any given $\epsilon, \delta \in (0, 1)$. The canonical form adopted in most such learning theories is provided next,

Definition 4. **Canonical Forms**

Generalization Bound: For a given hypothesis class $\mathcal{H}$, training set $S \sim \mathcal{D}^n$, and $\epsilon, \delta \in (0, 1)$, a

typical generalization bound adopts the following form:

$$\text{with probability at least } 1-\delta \text{ we have, } \forall h \in \mathcal{H} \quad R(h) \leq R_S(h) + \underbrace{O\left(p_1\left(\frac{1}{\delta}, n, |\mathcal{H}|\right)\right)}_{\text{confidence term}}. \tag{10}$$

A direct implication of Equation (10) and uniform convergence property is the sample complexity provided next,

Sample Complexity: A hypothesis class $\mathcal{H}$ is PAC learnable using ERM-based algorithms with a training set S of sample complexity $n \geq p_2(\frac{1}{\epsilon}, \frac{1}{\delta}, |\mathcal{H}|)$.

The exact forms of the confidence terms and the sample complexity term depends on the way the complexity of the hypothesis class is captured. A brief survey of some the popular theories used to capture the complexity of the hypothesis class is provided in Table 9.

Note that Table 9 provides a very brief highlight of some of the popular traditional learning theories. For a more detailed coverage of advanced learning theories, please see References [132, 155]. In addition, there are a few alternative theories that use different learning mechanisms like mistake bounds [132], learning by distance [9], and statistical query learning [94]. Most such settings are adapted for very specific learning tasks and have some connections to the PAC learning theory. Interested readers are directed to the above references for further details.

Computation complexity: Another aspect of the PAC learning theory is the computational complexity of the algorithm. Note that there can be several algorithms to obtain the same solution. For example, to sort an array of numbers both merge sort and binary sort will provide the sorted output. Although the outcome (solution) of the algorithm may be the same, the worst-case computation complexity of each algorithm is different, i.e., binary sort $O(n^2)$ and merge sort $O(nlogn)$. PAC theory captures the efficiency of an algorithm in terms of its computation complexity. However, the computation times of any algorithm is machine dependent. To decouple such dependencies most computation complexity analyses assumes an underlying abstract machine, like a Turing Machine over reals, and so on, and provides a comparative machine independent computational analysis in big-O notation. For example, an $O(nlogn)$ implementation of the merge sort algorithm would mean the actual runtime in seconds on any machine would follow (see Reference [155]): "there exist constants c and n_0, which can depend on the actual machine, such that, for any value of $n > n_0$, the runtime in seconds of sorting any n items will be at most $cnlog(n)$."

A brief coverage of the computation times for several popular algorithms in big-O notation is provided in Table 2. For a more in-depth discussion on several computational models readers are directed to Reference [94].

As seen in this section, most popular traditional learning theories mainly target the sample complexity and in turn the generalization capability of an algorithm to learn a hypothesis class followed by the computation complexity of the algorithm to learn such a hypothesis class. From a resource constrained perspective although the computational (processing power) aspect of an algorithm is (asymptotically) handled in such theories, the interplay between computation vs. sample complexity is disjunctive. Even so, most of the existing methodologies adopted for resource constrained machine learning (discussed in sections Sections 3.3, 3.2.1, 3.2.2, 3.2.4, and 3.2.3) follow this learning paradigm.

4.2.2 Resource-Constrained Learning Theories. Although the PAC learning theory is the dominant theory behind majority of the ML algorithms, the notion of sample complexity and resource complexity (computation, memory, communication bandwidth, power, etc.) is disjunctive in such theories. In fact, the works in References [40, 152] raise some critical questions about the adequacy

Table 9. Popular Traditional Learning Theories

Theory		Confidence term $p_1(\frac{1}{\delta}, n, \lvert\mathcal{H}\rvert)$	Sample Complexity $p_2(\frac{1}{\epsilon}, \frac{1}{\delta}, \lvert\mathcal{H}\rvert)$	Additional Remarks
Finite Hypothesis ($\lvert\mathcal{H}\rvert < \infty$) Realizable ($\min_{h\in\mathcal{H}} R(h) = 0$)	PAC Bound [177]	$\frac{log\lvert\mathcal{H}\rvert + log\frac{1}{\delta}}{n}$	$\frac{1}{\epsilon}(log\lvert\mathcal{H}\rvert + log\frac{1}{\delta})$	
Non-Realizable ($\min_{h\in\mathcal{H}} R(h) \neq 0$)	Agnostic PAC [177]	$\sqrt{\frac{log\lvert\mathcal{H}\rvert + log\frac{2}{\delta}}{2n}}$	$\frac{log\lvert\mathcal{H}\rvert + log\frac{2}{\delta}}{\epsilon^2}$	
Countably Infinite ($\mathcal{H} = \cup_{n\in N} h_n$)	MDL [155]	$\sqrt{\frac{\lvert d(h)\rvert + log\frac{2}{\delta}}{2n}}$	$\frac{log\lvert d(h)\rvert + log\frac{2}{\delta}}{\epsilon^2}$	$d(h)$ = minimum description length of hypothesis.
Infinite Hypothesis	VC (dimension) Theory [182]	– $\sqrt{2\frac{log\mathcal{G}_{\mathcal{H}}(n) + log\frac{2}{\delta}}{n}}$ – $\sqrt{2\frac{dlog\frac{2en}{d} + log\frac{2}{\delta}}{n}}$ (Sauer's Lemma)	$\frac{log\mathcal{G}_{\mathcal{H}}(n) + log\frac{2}{\delta}}{\epsilon^2}$	– Growth Function $\mathcal{G}_{\mathcal{H}}(n) = \sup_{z_1\ldots z_n\sim\mathcal{D}^n} \lvert\mathcal{H}_{z_1\ldots z_n}\rvert$. – VC dim, $d = \max_{n\in N:\mathcal{G}_{\mathcal{H}}(n)=2^n} n, \quad z_i = (\mathbf{x}_i, y_i)$.
	VC (Entropy) Theory [182]	$\sqrt{2\frac{logE_D[N(\mathcal{H},2n)] + log\frac{2}{\delta}}{n}}$	$\frac{logE_D[N(\mathcal{H},2n)] + log\frac{2}{\delta}}{\epsilon^2}$	Size of function class $N(\mathcal{H}, n) = \lvert\mathcal{H}_{z_1\ldots z_n}\rvert$.
	Covering Number [197]	$\sqrt{\frac{log(4\Gamma_1(2n,\varepsilon/8,\mathcal{F}) + log(\frac{1}{\delta}))}{2n}}$	$\frac{log(4\Gamma_1(2n,\varepsilon/8,\mathcal{F}) + log(\frac{1}{\delta}))}{\epsilon^2}$	– Covering number $\Gamma_p(n, \varepsilon, \mathcal{H}) = \max\{N_{in}(\varepsilon, \mathcal{H}, \lvert\lvert\cdot\rvert\rvert_{p,x}) \lvert \mathbf{x}\in\mathcal{X}\}$ – $N_{in}(\varepsilon, \mathcal{H}, \lvert\lvert\cdot\rvert\rvert_{p,x})$ smallest cardinality of internal ε cover of $\mathcal{H}$ with $\lvert\lvert g\rvert\rvert_{p,z} = (\frac{1}{n}\sum_{i=1}^{n}\lvert g(z_i)\rvert^p)^{\frac{1}{p}}$.
	Radamacher Complexity [6]	– $2\mathcal{R}_n(\mathcal{H}) + \sqrt{\frac{log\frac{1}{\delta}}{2n}}$ – or, $2\mathcal{R}_S(\mathcal{H}) + 3\sqrt{\frac{log\frac{2}{\delta}}{2n}}$		– Empirical Radamacher Complexity, $\mathcal{R}_S(\mathcal{H}) = [\sup_{\substack{h\in\mathcal{H}\\ \sigma}} \frac{\sum_{i=1}^{n}\sigma_i h(\mathbf{x}_i)}{n}]$ – Radamacher Complexity, $\mathcal{R}_n(\mathcal{H}) = \underset{S\in\mathcal{D}^n}{R_S(\mathcal{H})}$. For equivalence with, Gaussian complexity, Maximum Discrepancy, etc. see [6].
	Algorithm Stability [132]	$\beta + (2m\beta + M)\sqrt{\frac{log(\frac{1}{\delta})}{2m}}$		– Applies to β stable algorithms $h_S = A(S)$. – A is uniformly β stable if $\forall(\mathbf{x}, y) \in \mathcal{X}\times\{-1, +1\}$ $\lvert l(y, h_S(\mathbf{x})) - l(y, h'_S(\mathbf{x}))\rvert \leq \beta; S, S' \sim \mathcal{D}^n$.
	Compression Bounds [155]	$\sqrt{R_{S'}(A(S))\frac{4klog(m/\delta)}{m}} + \frac{8klog(m/\delta)}{m}$		– $S' = S_I$ where $\exists$ an index set I and a compression scheme $B : S_I \rightarrow \mathcal{H}$ $s.t.$ $A(S) = B(S_I)$ with $\lvert S\rvert > 2\lvert S_I\rvert$. – The bound is on the Algorithm A's output in terms of error on the hold-out set S'.

of the PAC learning framework for designing ML algorithms with limited computation complexity. These questions have led to a substantial amount of work modifying the PAC learning paradigm to provide an explicit tradeoff between sample and computation complexity. Good surveys of sample-computation complexity theory can be found in References [33, 34, 40, 46, 121, 154, 156, 160] and is not the main focus of this article. Rather, in this category we target the more exclusive class of literature that additionally captures the effect of space complexity on the sample complexity for learnability. There is limited research that provides an explicit characterization of such an interplay for any generic algorithm. Most such works modify the traditional assumption of i.i.d. data being presented in a batch or streaming fashion and introduces a specific protocol of data observability. Such theories limit the memory/space footprint through this restricted data observability. These advanced theories provide a platform to utilize existing computationally efficient algorithms under memory constrained settings to build machine learning models with strong error guarantees. We discuss a few examples of such work next.

A seminal work in this line can be found in Reference [8], where the authors introduce a new protocol for data observation called **Restricted focus of attention (RFA)**. This modified protocol is formalized through a projection (or focusing) function and limits the algorithm's memory (space) and computation footprint through selective observation of the available samples' attributes/features. The authors introduce a new notion of k-RFA or (k-weak-RFA) learnability, where k is the number of observed bits (or features) per sample and provides a framework to analyze the sample complexity of any learning algorithm under such memory constrained observation. The interplay between space and sample complexity of any algorithm designed in this RFA framework is provided in Table 10. Several follow-up works target a class of problems like l_1/l_2–linear regression, support vector regression, and so on, in a similar RFA framework and modified the projection function providing lower sample complexity [12, 69, 88, 89]. They also provide computationally efficient algorithms for these specific methods.

Another approach to characterize the sample vs. space complexity is provided in Reference [164]. Here the authors introduce a memory efficient streaming data observation protocol and utilizes the **statistical query- (SQ)**-based learning paradigm (originally introduced in Reference [94]) to characterize the sample vs. space complexity for learnability of finite hypothesis classes. Allowing the SQ algorithm for improper learning, the authors justify applicability to infinite function classes through an ϵ - cover under some metric. However, such extensions have not been provided in the article. One major advantage of this approach is that it opens up the gamut of machine learning algorithms developed in the SQ paradigm for resource constrained settings [47, 48]. As an example, the authors illustrate the applicability of their theory for designing resource efficient k-sparse linear regression algorithm following References [47, 48]. Additional details on the space/sample complexity under this framework is provided in Table 10.

More recently, References [134, 135] introduced a graph-based approach to model the version space of a hypothesis class in the form of a *hypothesis graph*. The authors introduced a notion of *d-mixing* of the hypothesis graph as a measure of (un)-learnability in bounded memory settings. This notion of mixing was formalized as a complexity measure in Reference [134] and further utilized to show the un-learnability property of most generic neural network architectures. In another line of work a similar (yet complimentary) notion of *separability* was introduced in Reference [136]. Rather than showing the negative (unlearnability) results under bounded memory settings; this framework was used to characterize the lower bound sample vs. space complexity for learnability of a hypothesis class using an (n, b, δ, ϵ)-bounded memory algorithm. Here, the data generating distribution is assumed to be uniform in the domain, n = number of training samples, b = bits of memory, δ, ϵ = confidence, tolerance values for PAC learnability. However, the framework

Table 10. Resource Constrained Learning Theories

Theory	Sample Complexity	Space Complexity	Additional Remarks
$k-$RFA ([8] Corollary 4.3)	$max\{\frac{4r^2}{\epsilon} log \frac{2r}{\delta}, \frac{8r^2 VCdim(\mathcal{H})}{\epsilon} log \frac{13r}{\epsilon}\}$	$O(k)$ per sample	– $\mathcal{H} = \Psi(\mathcal{F}_i)_{i=1}^r$, where Composition function $\Psi(\mathcal{F}_i)_{i=1}^r = \{\psi(f_1, \ldots, f_r) \vert f_i \in \mathcal{F}\}$ defines an ensemble of functions. – $\mathcal{F}$ is PAC learnable over domain X_k. $k < d$ is a pre-selected subset of features. – r number of functions used in the ensemble function Ψ. This is user-defined. – Captures the per-sample space complexity and not the overall algorithm space complexity.
Statistical Query ([164] Theorem 7)	$O(\frac{\lceil m_0/klog\vert\mathcal{H}\vert}{\tau^2}(loglog\vert\mathcal{H}\vert + logm_0 + log(1/\delta)))$	$O(log\vert\mathcal{H}\vert(logm_0 + loglog(1/\tau)) + klog(1/\tau))$ per state variable	– Assumption : $\mathcal{H}$ is $(\epsilon, 0)-$ learnable with m_0 statistical queries of tolerance τ. – k = user defined tradeoff with inverse-dependence on sample complexity and direct dependence on space complexity. – $\vert\mathcal{H}\vert$ = size of the finite hypothesis class of probability distributions. – Following this framework provides a k-sparse linear regression implementation with sample complexity $\tilde{O}(\frac{nk^8 log(1/\delta)}{\epsilon^4})$ and space complexity $O(klog^2(\frac{d}{\epsilon}))$ bits. $\tilde{O}(\cdot)$ is big - O, which additionally hides the log terms.
$\mathcal{H}-$ graph separability ([136] Theorem 7)	$\frac{k}{\alpha} log \frac{\vert\mathcal{H}\vert}{\alpha^2}$	$log \frac{\vert\mathcal{H}\vert}{\alpha^2} + log \frac{k}{\alpha}$ bits	– Assumption : $\mathcal{H}-$ graph is $(\alpha, \epsilon)-$ separable (see [164] for definition). – $\mathcal{H}$ is PAC learnable for given $(\epsilon, \delta = e^{-k\alpha^2/8} + e^{-2k\epsilon^2})$. – Limited to uniform distributions and targets sample complexity lower bounds.
Hide-n-Seek ([157] Theorem 3)	$\Omega(\max\{(d/\rho b), 1/\rho^2\})$	$O(T(b + n_t d))$	– Applicable only to estimation problems like PCA, SVD, CCA, and so on, falling in the hide-n-seek framework. – b = space complexity of intermediate results. – n_t = size of a mini-batch (of i.i.d. samples with dimension d) at $t \in T$ iteration. – T = number of epochs for the iterative algorithm.

supports a very limited set of machine learning algorithms and its applicability for modifying popular machine learning algorithms have not yet been shown.

Finally, Reference [146] proposes a new protocol for data access called *branching program* and translates the learning algorithm under resource (memory) constraints in the form of a matrix (as opposed to a graph in Reference [134]). The authors build a connection between the stability of the matrix norm (in the form of an upper bound on its maximum singular value) and the learnability of the hypothesis class with limited memory. This work provides the interplay between the memory-size and minimum number of samples required for exact learning of the concept class. Reference [7] extended this work for the class of problems where the sample space of tests is smaller than the sample space of inputs. Reference [54] further improves upon the bounds proposed in References [7, 146]. However, most existing work in this framework focus on analyzing the learnability of a problem, rather than providing an algorithm guaranteeing learnability for resource constrained settings.

In addition to the above theories there are a few research works that target a niche yet interesting class of machine learning problems like hypothesis testing [43, 72, 73, 101, 108] or function estimation [5, 74, 133]. However, their extension to any generic loss function has not been provided and is non-trivial. Hence, we do not delve into the details of such settings. However, one specific research on estimating biased coordinates [157] needs special consideration, mainly because it covers a set of very interesting problems like principal component analysis, singular value decomposition, correlation analysis, and so on. This work adopts an information theory centric general framework to handle most memory constrained estimation problems. The authors introduce a generic (b, n_t, T) protocol of data observability for most iterative mini-batch-based algorithms. Here, b = space complexity of intermediate results, n_t = size of a mini-batch (of i.i.d. samples with dimension d) at $t \in T$ iteration, where T = number of epochs for the iterative algorithm. In a loose sense the overall space complexity including the data and intermediate results become $O(T(b + n_t d))$. Under such a (b, n_t, T) protocol the authors casts most estimation problems like sparse PCA, Covariance estimation, correlation analysis, and so on, into a "*hide-and-seek*" problem. Using this framework the authors explore the limitations in memory constrained settings and provide the sample complexity for good estimation with high probability (see Table 10). However, although this provides a framework to analyze specific estimation problems, there are no guidelines on how to cast/modify existing algorithms to optimize the space vs. sample complexities within this framework.

4.3 Challenges in Resource-efficient Theoretical Research

As presented in the above section there are some major challenges underlying the resource efficient theories discussed in Section 4.2.2.

(1) Most of the new theories are mainly developed to analyze the un-learnability of a hypothesis class $\mathcal{H}$ under resource constraints. Adapting such frameworks to develop resource efficient algorithms guaranteeing the learnability of $\mathcal{H}$ is needed, although there are a few theoretical frameworks like References [8, 164] that provide guidelines toward developing resource efficient algorithms. However, the underlying assumptions rule out a wide range of hypothesis classes. Showing the practicality of such assumptions toward developing a wide range of machine learning algorithms is a huge challenge in such existing frameworks.

(2) In addition, as shown in Table 10 most of the existing theories deal with a specific aspect of the space-complexity. For example, Reference [8] mainly considers the per-sample space complexity while Reference [164] considers the space complexity of the intermediate state representation. A more comprehensive analysis of the overall space/computation complexity of the algorithm $A(S)$ needs to be developed.

(3) A general limitation of most of the theoretical frameworks is the limited empirical analysis of the algorithms designed using these frameworks. For example, most of the analyses are asymptotic in nature. How such mathematical expressions translate in practicality is still an open problem.
(4) Most of the existing theories introduces error guarantees in terms of the hypothesis class. Selecting the optimal model through hyperparameter optimization, model selection routines in a resource constrained setting and guaranteeing the correctness of the selected model is missing.
(5) Finally, a comparison between the frameworks introduced in Table 10 is missing.

5 DISCUSSION

The previous sections provide a comprehensive look at the current state of on-device learning from an algorithm and theory perspective. In this section, we provide a brief summary of these findings and elaborate on the research and development challenges facing the adoption of an edge learning paradigm. We also highlight the effort needed for on-device learning using a few typical edge-learning use cases and explain how research in the different areas (algorithms and theory) has certain advantages and disadvantages when it comes to their usability.

5.1 Summary of the Current State-of-the-art in On-device Learning

We begin by briefly summarizing our findings from Sections 3 and 4. If you have read those sections, then you can skip this summary and move directly to Section 5.2.

Algorithm Research (details in Section 3): Algorithm research mainly targets the computational aspects of model building under limited resource settings. The main goal is to design optimized machine learning algorithms that best satisfies a surrogate software-centric resource constraint. Such surrogate measures are designed to approximate the hardware constraints through asymptotic analysis (Section 3.1.1), resource profiling (Section 3.1.2), or resource modeling (Section 3.1.3). For a given software-centric resource constraint, the state-of-art algorithm designs adopt one of the following approaches:

(1) Lightweight ML Algorithms (see Section 3.2.1): This approach utilizes already available algorithms with low resource footprints. There are no additional modifications for resource constrained model building. As such, for cases where the available device's resources are smaller than the resource footprint of the selected lightweight algorithm, this approach will fail. Additionally, in most cases the lightweight ML algorithms result in models with low complexity that may fail to fully capture the underlying process.
(2) Reducing Model complexity (see Section 3.2.2): This approach controls the size (memory footprint) and computation complexity of the machine learning algorithm by adding additional constraints on to the model architecture (e.g., by selecting a smaller hypothesis class). For a pre-specified constrained model architecture motivated by the available resource constraints, this approach adopts traditional optimization routines. Apart from model building, this is one of the dominant approaches for deploying resource efficient models for model inference. Compared to the lightweight ML algorithms approach, model complexity reduction techniques can accommodate a broader class of ML algorithms and can more effectively capture the underlying process.
(3) Modifying optimization routines (see Section 3.2.3): This approach designs specific optimization routines for resource efficient model building. Here the resource constraints are incorporated during the model building (training) phase. Note that as opposed to the

previous technique of limiting the model architectures beforehand, this approach can adapt the optimization routines to fit the resource constraints for any given model architecture (hypothesis class). In certain cases, this approach can also dynamically modify the architecture to fit the resource constraints. Although this approach provides a wider choice of the class of models, the design process is still tied to a specific problem type (classification, regression, etc.) and adopted method/loss function (linear regression, ridge regression for regression problems).

(4) Data Compression (see Section 3.2.4): Rather than constraining the model size/complexity, this approach targets building models on compressed data. The goal is to limit the memory usage via reduced data storage and computation through fixed per-sample computation cost. In addition a more generic approach includes adopting advanced learning settings that accommodates algorithms with smaller sample complexity. However, this is a broader research topic and is not just limited to on-device learning. A detailed analysis of these approaches have been delegated to other existing surveys.

(5) New protocols for data observation (see Section 3.2.5): This approach completely changes the traditional data observation protocol (like availability of i.i.d. data in batch or online settings) and builds resource efficient models under limited data observation. These approaches are guided by an underlying resource constrained learning theory (discussed in Section 4.2.2) that captures the interplay between resource constraints and the goodness of the model in terms of the generalization capacity. Additionally, compared to the above approaches, this framework provides a generic mechanism to design resource constrained algorithms for a wider range of learning problems applicable to any method/loss function targeting that problem type.

Obviously one of the major challenges in this research is proper software-centric characterization of the hardware constraints and appropriately using this characterization for better metric designs. Some other important challenges include applicability to a wider range of algorithms and dynamic resource budgeting. A more detailed discussion is available in Section 3.4.

Theory Research (details in Section 4): Research into the theory underlying on-device learning focuses mainly on developing frameworks to analyze the statistical aspects (i.e., error guarantees) of a designed algorithm with or without associated resource constraints. There are two broad categories into which most of the existing resource constrained algorithms can be categorized,

(1) Traditional Learning Theories (see Section 4.2.1): Most of the existing resource constrained algorithm design (summarized in (1)–(4) above) follow this traditional machine learning theory. A limitation of this approach is that such theories are built mainly for analyzing the error guarantees of the algorithm used for model estimation. The effect of resource constraints on the generalization capability of the algorithm is not directly addressed through such theories. For example, algorithms developed using the approach of *reducing the model complexity* typically adopts a two step approach. First the size of the hypothesis class is constrained *a priori* to incorporate the resource constraints. Next, an algorithm is designed guaranteeing the best-in-class model within that hypothesis class. The direct interplay between the error guarantees and resource constraints is missing in such theoretical frameworks.

(2) Resource constrained learning theories (see Section 4.2.2): Modern research has shown that it may be impossible to learn a hypothesis class under resource constrained settings. To circumvent such inconsistencies in traditional learning theories, newer resource constraint learning theories have been developed. Such theories provide learning guarantees in light of

the resource constraints imposed by the device. The algorithms designed using the approach summarized in point 5 above follow these newer learning theories. Although such theory motivated design provides a generic framework through which algorithms can be designed for a wide range of learning problems for any loss function addressing the problem type, till date, very few algorithms based on these theories have been developed.

Overall, the newer resource constrained theory research provides a generic framework for designing algorithms with error guarantees under resource constrained settings that apply to a broader range of problems. However, currently, very few algorithms have been developed that utilize these frameworks. Developing additional algorithms in such advanced frameworks need significant effort. Also, the application of such theories to a complete ML pipeline including hyperparameter optimization, data preprocessing, and so on, has not yet been addressed.

5.2 Research and Development Challenges

A study of the current state of the art in on-device learning also provides an understanding of the existing challenges in the field that prevent its adoption as a real alternative to cloud-based solutions. In this section we identify a few directions for research that will allow us to build learning algorithms that run on the edge. As before, we limit our discussion to the algorithm and theory levels.

Algorithm: The challenges facing the research & development of resource constrained optimized algorithms falls into three areas:

(1) Software-centric Resource Constraints: A correct characterization of the software-centric resource constraints that best approximates the hardware resources is absolutely critical to developing resource constrained algorithms, because the optimal algorithm design is very specific to a particular device (i.e., computational model) and its available resources.
(2) Understanding how traditional ML algorithms can contribute to edge learning: Majority of current on-device algorithm research focuses on adapting modern deep learning approaches like CNN, DNN to run on the edge. Very little focus has been given to traditional machine learning methods. Traditional machine learning methods are of huge interest for building edge learning capability especially when memory is limited (for devices with megabytes or even kilobytes of RAM) and computational power is low. These are precisely the areas where modern deep architectures are wholly unsuited due to their out-sized memory and compute requirements. Hence, there is a need to explore the traditional machine learning approaches for learning on-device. In addition, designing advanced learning algorithms that requires small training datasets to achieve high test accuracies is a huge challenge. Addressing this capability can significantly improve the resource footprint (i.e., low memory footprint to store the training data).
(3) Dynamic resource budget: Most existing research assume the availability of dedicated resources while training their edge-learning algorithms. However, for many applications such an assumption is invalid. For example, in mobile phones an application's priority may change [45] based on user's actions. Hence, there is a need for algorithm research incorporating the accuracy–latency tradeoff under dynamic resource constraints.

For building edge-learning capability for use-cases like object detection, user identification, and so on, specific to products like., mobile phones, refrigerator, and so on; significant groundwork needs to be laid in terms of a complete analysis of existing approaches that cater to the specific requirements of the use-cases and the products. This analysis also forms the basis for extending existing approaches and developing newer algorithms in those cases where current methods fail to perform satisfactorily.

Theory: The research on algorithms developed using (extensions of) advanced theoretical frameworks provides a mechanism to characterize the performance (error) guarantees of a model in resource constrained settings. The effort on such research needs careful consideration of the following points,

(1) Analysis of Resource Constraints: Although advanced theories abstract resource constraints as a form of information bottleneck, a good understanding of how such abstractions practically impact a resource (say memory vs. computation) needs to be properly analyzed.
(2) Applicability: With the appropriate abstraction in place, applicability (of the underlying assumptions) of the theory for practical use-cases needs to be analyzed.
(3) Algorithm development: Once newer learning theories supporting resource constraints have been developed, there is need to develop new algorithms based on these new theoretical frameworks.
(4) Extending existing theories: The existing theories majorly cater to a very specific aspect of the ML pipeline (i.e., training an algorithm for a pre-specified hypothesis class). Extending this approach to the entire model building pipeline (e.g., Hyper-parameter optimization, data pre-processing, etc.) remains to be developed. Also, a majority of the underlying theories target the un-learnability of a problem. Extending such concepts to newer theories to develop practical algorithms needs to be developed.

Overall, the theory research provides a mechanism to characterize a wide range of edge-specific ML problems. However, extending research at this level needs significant effort as compared to the previous algorithmic approaches.

6 CONCLUSION

On-device learning has so far remained in the purview of academic research, but with the increasing number of smart devices and improved hardware, there is interest in performing learning on the device rather than in the cloud. In the industry, this interest is fueled mainly by hardware manufacturers promoting AI-specific chipsets that are optimized for certain mathematical operations, and startups providing ad hoc solutions to certain niche domains mostly in computer vision and IoT. Given this surge in interest and corresponding availability of edge hardware suitable for on-device learning, a comprehensive survey of the field from an algorithms and learning theory perspective sets the stage for both understanding the state of the art and for identifying open challenges and future avenues of research. However, on-device learning is an expansive field with connections to a large number of related topics in AI and machine learning including online learning, model adaptation, one/few-shot learning, resource-constrained learning, and so on, to name just a few. Covering such a large number of research topics in a single survey is impractical but, at the same time, ignoring the work that has been done in these areas leaves significant gaps in any comparison of approaches. This survey finds a middle ground by reformulating the problem of on-device learning as resource-constrained learning where the resources are compute and memory. This reformulation allows tools, techniques, and algorithms from a wide variety of research areas to be compared equitably.

We limited the survey to learning on single devices with the understanding that the ideas discussed can be extended in a normal fashion to the distributed setting via an additional constraint based on communication latency. We also focused the survey on the algorithmic and theoretical aspects of on-device learning leaving out the effects of the systems level (hardware and libraries). This choice was deliberate and allowed us to separate out the algorithmic aspects of on-device learning from implementation and hardware choices. This distinction also allows us to identify challenges and future research that can be applied to a variety of systems.

Based on the reformulation of on-device learning as resource constrained learning, the survey found that there are a number of areas where more research and development is needed.

At the algorithmic level, it is clear that current efforts are mainly targeted at either utilizing already lightweight machine learning algorithms or modifying existing algorithms in ways that reduce resource utilization. There are a number of challenges we identified in the algorithm space including the need for decoupling algorithms from hardware constraints, designing effective loss functions and metrics that capture resource constraints, an expanded focus on traditional as well as advanced ML algorithms with low sample complexity in addition to the current work on DNNs, and dealing with situations where the resource budget is dynamic rather than static. In addition, improved methods for model profiling are needed to more accurately calculate an algorithm's resource consumption. Current approaches to such measurements are abstract and focus on applying software engineering principles such as asymptotic analysis or low-level measures like FLOPS or MACs. None of these approaches give a holistic idea of resource requirements and in many cases represent an insignificant portion of the total resources required by the system during learning.

Finally, current research in the field of learning theory for resource constrained algorithms is focused on the un-learnability of an algorithm under resource constraints. The natural step forward is to identify techniques that can instead provide guarantees on the learnability of an algorithm and the associated estimation error. Existing theoretical techniques also mainly focus on the space(memory) complexity of these algorithms and not their compute requirements. Even in cases where an ideal hypothesis class can be identified that satisfies resource constraints, further work is needed to select the optimal model from within that class.

REFERENCES

[1] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2016. Fathom: Reference workloads for modern deep learning methods. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'16)*. IEEE, 1–10.

[2] Furqan Alam, Rashid Mehmood, Iyad Katib, and Aiiad Albeshri. 2016. Analysis of eight data mining algorithms for smarter Internet of Things (IoT). *Proc. Comput. Sci.* 98 (2016), 437–442.

[3] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. *Advances in Neural Information Processing Systems* 30 (2017), 1709–1720.

[4] Zeyuan Allen-Zhu and Yuanzhi Li. 2017. First efficient convergence for streaming k-pca: A global, gap-free, and near-optimal rate. In *Proceedings of the 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS'17)*. IEEE, 487–492.

[5] Noga Alon, Yossi Matias, and Mario Szegedy. 1999. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.* 58, 1 (1999), 137–147.

[6] Peter L. Bartlett and Shahar Mendelson. 2002. Rademacher and Gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.* 3 (Nov. 2002), 463–482.

[7] Paul Beame, Shayan Oveis Gharan, and Xin Yang. 2017. Time-space tradeoffs for learning from small test spaces: Learning low degree polynomial functions. arXiv:1708.02640. Retrieved from https://arxiv.orb/abs/1708.02640.

[8] Shai Ben-David and Eli Dichterman. 1998. Learning with restricted focus of attention. *J. Comput. System Sci.* 56, 3 (1998), 277–298.

[9] Shai Ben-David, Alon Itai, and Eyal Kushilevitz. 1990. Learning by distances. In *Proceedings of the Annual Conference on Learning Theory (COLT'90)*. 232–245.

[10] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. 2018. Benchmark analysis of representative deep neural network architectures. *IEEE Access* (2018).

[11] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*. PMLR, 527–536.

[12] Brian Bullins, Elad Hazan, and Tomer Koren. 2016. The limits of learning with missing data. In *Advances in Neural Information Processing Systems*. 3495–3503.

[13] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. 2017. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. In *Asian Conference on Machine Learning*. PMLR, 622–637.

[14] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce memory, not parameters for efficient on-device learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS'20)*.

[15] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. arXiv:1812.00332. Retrieved from arxiv.org/abs/1812.00332.
[16] Léopold Cambier, Anahita Bhiwandiwalla, Ting Gong, Mehran Nekuii, Oguz H. Elibol, and Hanlin Tang. 2020. Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks. arXiv:2001.05674. Retrieved from arxiv.org/abs/2001.05674.
[17] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. 2016. An analysis of deep neural network models for practical applications. arXiv:1605.07678. Retrieved from https://arxiv.org/abs/1605.07678.
[18] Nicolo Cesa-Bianchi, Shai Shalev-Shwartz, and Ohad Shamir. 2011. Efficient learning with partially observed attributes. *J. Mach. Learn. Res.* 12 (Oct. 2011), 2857–2878.
[19] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. 2009. Semi-supervised learning. *IEEE Trans. Neural Netw.* 20, 3 (2009), 542–542.
[20] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. arXiv:1604.06174. Retrieved from https://arxiv.org/abs/1604.06174.
[21] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *Proceedings of the International Conference on Machine Learning*. 2285–2294.
[22] Xie Chen, Xunying Liu, Yongqiang Wang, Mark J. F. Gales, and Philip C. Woodland. 2016. Efficient training and evaluation of recurrent neural network language models for automatic speech recognition. *IEEE/ACM Trans. Aud. Speech Lang. Process.* 24, 11 (2016), 2146–2157.
[23] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circ.* 52, 1 (2017), 127–138.
[24] An-Chieh Cheng, Jin-Dong Dong, Chi-Hung Hsu, Shu-Huan Chang, Min Sun, Shih-Chieh Chang, Jia-Yu Pan, Yu-TingChen, Wei Wei, and Da-Cheng Juan. 2018. Searching toward pareto-optimal device-aware neural architectures. In *Proceedings of the International Conference on Computer-Aided Design*. 1–7.
[25] Vladimir Cherkassky and Filip M. Mulier. 2007. *Learning from Data: Concepts, Theory, and Methods*. John Wiley & Sons.
[26] H. Choi, W. P. Burleson, and D. S. Phatak. 1993. Fixed-point roundoff error analysis of large feedforward neural networks. In *Proceedings of the 1993 International Conference on Neural Networks (IJCNN'93)*, Vol. 2. IEEE, 1947–1950.
[27] Cheng-Tao Chu, Sang K. Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y. Ng. 2007. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems*. 281–288.
[28] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. DAWNBench: An end-to-end deep learning benchmark and competition. *Training* 100, 101 (2017), 102.
[29] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*. 3123–3131.
[30] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. arXiv:1602.02830. Retrieved from https://arxiv.org/abs/1602.02830.
[31] Koby Crammer, Alex Kulesza, and Mark Dredze. 2013. Adaptive regularization of weight vectors. *Mach. Learn.* 91, 2 (2013), 155–187.
[32] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. 2019. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 11398–11407.
[33] Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. 2013. From average case complexity to improper learning complexity. arXiv:1311.2272. Retrieved from https://arxiv.org/abs/1311.2272.
[34] Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. 2014. From average case complexity to improper learning complexity. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. 441–448.
[35] Saumitro Dasgupta and David Gschwend. Netscope CNN Analyzer. Retrieved December 22, 2020 from https://dgschwend.github.io/netscope/quickstart.html.
[36] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. 2017. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 561–574.
[37] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R. Aberger, Kunle Olukotun, and Christopher Ré. 2018. High-accuracy low-precision training. arXiv:1803.03383. Retrieved from https://arxiv.org/abs/1803.03383.
[38] Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. 2015. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in Neural Information Processing Systems*. 2674–2682.

[39] 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*. 1223–1231.

[40] Scott E. Decatur, Oded Goldreich, and Dana Ron. 2000. Computational sample complexity. *SIAM J. Comput.* 29, 3 (2000), 854–879.

[41] Swarnava Dey, Arijit Mukherjee, and Arpan Pal. 2019. Embedded deep inference in practice: Case for model partitioning. In *Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems*. 25–30.

[42] Sauptik Dhar, Vladimir Cherkassky, and Mohak Shah. 2019. Multiclass learning from contradictions. In *Advances in Neural Information Processing Systems*. 8400–8410.

[43] Kimon Drakopoulos, Asuman Ozdaglar, and John N. Tsitsiklis. 2013. On learning with finite memory. *IEEE Trans. Inf. Theory* 59, 10 (2013), 6859–6872.

[44] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme model compression. arXiv:cs.LG/2004.07320. Retrieved from https://arxiv.org/abs/2004.07320.

[45] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 115–127.

[46] Vitaly Feldman et al. 2007. *Efficiency and Computational Limitations of Learning Algorithms*. Vol. 68.

[47] Vitaly Feldman, Elena Grigorescu, Lev Reyzin, Santosh Vempala, and Ying Xiao. 2013. Statistical algorithms and a lower bound for detecting planted cliques. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*. ACM, 655–664.

[48] Vitaly Feldman, Cristóbal Guzmán, and Santosh Vempala. 2017. Statistical query algorithms for mean vector estimation and stochastic convex optimization. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1265–1277.

[49] Jiashi Feng and Trevor Darrell. 2015. Learning the structure of deep convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 2749–2757.

[50] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.* 15, 1 (2014), 3133–3181.

[51] M Fernández-Delgado, M. S. Sirsat, Eva Cernadas, Sadi Alawadi, Senén Barro, and Manuel Febrero-Bande. 2019. An extensive experimental survey of regression methods. *Neural Netw.* 111 (2019), 11–34.

[52] Venkata Gandikota, Daniel Kane, Raj Kumar Maity, and Arya Mazumdar. 2021. vqsgd: Vector quantized stochastic gradient descent. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2197–2205.

[53] Wanling Gao, Jianfeng Zhan, Lei Wang, Chunjie Luo, Daoyi Zheng, Rui Ren, Chen Zheng, Gang Lu, Jingwei Li, Zheng Cao, et al. 2018. BigDataBench: A dwarf-based big data and AI benchmark suite. arXiv:1802.08254. Retrieved from https://arxiv.orb/abs/1802.08254.

[54] Sumegha Garg, Ran Raz, and Avishay Tal. 2018. Extractor-based time-space lower bounds for learning. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 990–1002.

[55] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. 2020. Recent advances in open set recognition: A survey. *IEEE Trans. Pattern Anal. Mach. Intell* (2020), 1–1.

[56] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. 2018. Squeezenext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 1638–1647.

[57] Eugenio Gianniti, Li Zhang, and Danilo Ardagna. 2018. Performance prediction of gpu-based deep learning applications. In *Proceedings of the 2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'18)*. IEEE, 167–170.

[58] Daniel Golovin, D. Sculley, Brendan McMahan, and Michael Young. 2013. Large-scale learning with less ram via randomization. In *Proceedings of the International Conference on Machine Learning*. 325–333.

[59] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. arXiv:1412.6115. Retrieved from https://arxiv.org/abs/1412.6115.

[60] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv:1412.6572. Retrieved from https://arxiv.org/abs/1412.6572.

[61] Klaus Greff, Rupesh K. Srivastava, and Jürgen Schmidhuber. 2017. Highway and residual networks learn unrolled iterative estimation. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

[62] Alex Grubb and Drew Bagnell. 2012. Speedboost: Anytime prediction with uniform near-optimality. In *Artificial Intelligence and Statistics*. 458–466.

[63] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. 2016. Memory-efficient backpropagation through time. In *Advances in Neural Information Processing Systems*. 4125–4133.

[64] Renjie Gu, Shuo Yang, and Fan Wu. 2019. Distributed machine learning on mobile devices: A survey. arXiv:1909.08329. Retrieved from https://arxiv.org/abs/1909.08329.
[65] Yunhui Guo. 2018. A survey on methods and theories of quantized neural networks. arXiv:1808.04752. Retrieved from https://arxiv.org/abs/1808.04752.
[66] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the International Conference on Machine Learning*. 1737–1746.
[67] Karen Zita Haigh, Allan M. Mackay, Michael R. Cook, and Li G. Lin. 2015. *Machine Learning for Embedded Systems: A Case Study*. Technical Report.
[68] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv:1510.00149. Retrieved from https://arxiv.orb/abs/1510.00149.
[69] Elad Hazan and Tomer Koren. 2012. Linear regression with limited observation. arXiv:1206.4678. Retrieved from https://arxiv.org/abs/1206.4678.
[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[71] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 784–800.
[72] Martin Hellman. 1972. The effects of randomization on finite-memory decision schemes. *IEEE Trans. Inf. Theory* 18, 4 (1972), 499–502.
[73] Martin E. Hellman, Thomas M. Cover, et al. 1971. On memory saved by randomization. *Ann. Math. Stat.* 42, 3 (1971), 1075–1078.
[74] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. 1998. Computing on data streams. *External Mem. Algor.* 50 (1998), 107–118.
[75] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *Proceedings of the NIPS Deep Learning and Representation Learning Workshop*.
[76] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580. Retrieved from https://arxiv.org/abs/1207.0580.
[77] Markus Hoehfeld and Scott E. Fahlman. 1991. *Learning with Limited Numerical Precision Using the Cascade-correlation Algorithm*. Citeseer.
[78] J. L. Holt and Jenq-Neng Hwang. 1991. Finite precision error analysis for neural network learning. In *Proceedings of the 1st International Forum on Applications of Neural Networks to Power Systems*. IEEE, 237–241.
[79] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*. 1314–1324.
[80] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861. Retrieved from https://arxiv.org/abs/1704.04861.
[81] Cho-Jui Hsieh, Si Si, and Inderjit S. Dhillon. 2014. Fast prediction for large-scale kernel machines. In *Advances in Neural Information Processing Systems*. 3689–3697.
[82] Zhiming Hu, Ahmad Bisher Tarakji, Vishal Raheja, Caleb Phillips, Teng Wang, and Iqbal Mohomed. 2019. Deephome: Distributed inference with heterogeneous devices in the edge. In *Proceedings of the 3rd International Workshop on Deep Learning for Mobile Systems and Applications*. 13–18.
[83] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. 2018. Multi-scale dense networks for resource efficient image classification. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*.
[84] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q. Weinberger. 2018. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'18)*.
[85] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18, 1 (2017), 6869–6898.
[86] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size. arXiv:1602.07360. Retrieved from https://arxiv.org/abs/1602.07360.
[87] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*.

[88] Shinji Ito, Daisuke Hatano, Hanna Sumita, Akihiro Yabe, Takuro Fukunaga, Naonori Kakimura, and Ken-Ichi Kawarabayashi. 2017. Efficient sublinear-regret algorithms for online sparse linear regression with limited observation. In *Advances in Neural Information Processing Systems*. 4099–4108.

[89] Shinji Ito, Daisuke Hatano, Hanna Sumita, Akihiro Yabe, Takuro Fukunaga, Naonori Kakimura, and Ken-Ichi Kawarabayashi. 2018. Online regression with partial information: Generalization and linear projection. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*. 1599–1607.

[90] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[91] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. 2019. A survey of deep learning-based object detection. *IEEE Access* 7 (2019), 128837–128868.

[92] Cijo Jose, Prasoon Goyal, Parv Aggrwal, and Manik Varma. 2013. Local deep kernel learning for efficient non-linear svm prediction. In *Proceedings of the International Conference on Machine Learning*. 486–494.

[93] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. 2015. Reduced-precision strategies for bounded memory in deep neural nets. arXiv:1511.05236. Retrieved from https://arxiv.org/abs/1511.05236.

[94] Michael J. Kearns. 1990. *The Computational Complexity of Machine Learning*. MIT Press.

[95] Minje Kim and Paris Smaragdis. 2016. Bitwise neural networks. arXiv:1601.06071. Retrieved from https://arxiv.org/abs/1601.06071.

[96] Kuno Kollmann, K.-R. Riemschneider, and Hans Christoph Zeidler. 1996. On-chip backpropagation training using parallel stochastic bit streams. In *Proceedings of the 5th International Conference on Microelectronics for Neural Networks*. IEEE, 149–156.

[97] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[98] Vrushali Y. Kulkarni and Pradeep K. Sinha. 2012. Pruning of random forest classifiers: A survey and future directions. In *Proceedings of the 2012 International Conference on Data Science & Engineering (ICDSE'12)*. IEEE, 64–68.

[99] Ashish Kumar, Saurabh Goyal, and Manik Varma. 2017. Resource-efficient machine learning in 2 KB RAM for the Internet of Things. In *Proceedings of the International Conference on Machine Learning*. 1935–1944.

[100] Matt Kusner, Stephen Tyree, Kilian Weinberger, and Kunal Agrawal. 2014. Stochastic neighbor compression. In *Proceedings of the International Conference on Machine Learning*. 622–630.

[101] K. B. Lakshmanan and B. Chandrasekaran. 1979. Compound hypothesis testing with finite memory. *Inf. Contr.* 40, 2 (1979), 223–233.

[102] John Langford, Lihong Li, and Tong Zhang. 2009. Sparse online learning via truncated gradient. *J. Mach. Learn. Res.* 10 (Mar. 2009), 777–801.

[103] Quoc Le, Tamás Sarlós, and Alex Smola. 2013. Fastfood-approximating kernel expansions in loglinear time. In *Proceedings of the International Conference on Machine Learning*, Vol. 85.

[104] Scikit Learn. Decision Tree. Retrieved December 22, 2020 from http://scikit-learn.org/stable/modules/tree.html#complexity.

[105] Vadim Lebedev and Victor Lempitsky. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2554–2564.

[106] Jongmin Lee, Michael Stanley, Andreas Spanias, and Cihan Tepedelenlioglu. 2016. Integrating machine learning in embedded sensor systems for internet-of-things applications. In *Proceedings of the 2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT'16)*. IEEE, 290–294.

[107] Seulki Lee and Shahriar Nirjon. 2019. Neuro. ZERO: A zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 138–152.

[108] F. Leighton and Ronald Rivest. 1986. Estimating a probability using finite memory. *IEEE Trans. Inf. Theory* 32, 6 (1986), 733–742.

[109] Chun-Liang Li, Hsuan-Tien Lin, and Chi-Jen Lu. 2016. Rivalry of two families of algorithms for memory-restricted streaming pca. In *Artificial Intelligence and Statistics*. 473–481.

[110] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. 2017. Training quantized nets: A deeper understanding. In *Advances in Neural Information Processing Systems*. 5811–5821.

[111] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Sign. Process. Mag.* 37, 3 (2020), 50–60.

[112] Xiang Li, Tao Qin, Jian Yang, and Tieyan Liu. 2016. LightRNN: Memory and computation-efficient recurrent neural networks. In *Advances in Neural Information Processing Systems*. 4385–4393.

[113] Zheng Li and Christopher M. De Sa. 2019. Dimension-free bounds for low-precision training. In *Advances in Neural Information Processing Systems*. 11728–11738.

[114] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. 2000. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Mach. Learn.* 40, 3 (2000), 203–228.
[115] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. 2020. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Commun. Surv. Tutor.* 22, 3 (2020), 2031–2063.
[116] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv:1712.01887. Retrieved from https://arxiv.org/abs/1712.01887.
[117] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2015. Neural networks with few multiplications. arXiv:1510.03009. Retrieved from https://arxiv.org/abs/1510.03009.
[118] Ziheng Lin, Yan Gu, and Samarjit Chakraborty. 2010. Tuning machine-learning algorithms for battery-operated portable devices. In *Proceedings of the Asia Information Retrieval Symposium*. Springer, 502–513.
[119] Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. 2020. Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey. arXiv:2005.04275. Retrieved from https://arxiv.org/abs/2005.04275.
[120] Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta. 2017. Modeling the resource requirements of convolutional neural networks on mobile devices. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 1663–1671.
[121] Mario Lucic. 2017. *Computational and Statistical Tradeoffs via Data Summarization*. Ph.D. Dissertation. ETH Zurich.
[122] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis. 1996. A new method in neural network supervised training with imprecision. In *Proceedings of the 3rd International Conference on Electronics, Circuits, and Systems*, Vol. 1. IEEE, 287–290.
[123] Vicent Sanz Marco, Ben Taylor, Zheng Wang, and Yehia Elkhatib. 2020. Optimizing deep learning inference on embedded systems through adaptive model selection. *ACM Trans. Embed. Comput. Syst.* 19, 1 (2020), 1–28.
[124] Diana Marculescu, Dimitrios Stamoulis, and Ermao Cai. 2018. Hardware-aware machine learning: modeling and optimization. In *Proceedings of the International Conference on Computer-Aided Design*. 1–8.
[125] Prathamesh Mayekar and Himanshu Tyagi. 2020. RATQ: A universal fixed-length quantizer for stochastic optimization. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1399–1409.
[126] Ian McInerney, George A. Constantinides, and Eric C. Kerrigan. 2018. A survey of the implementation of linear model predictive control on fpgas. *IFAC-PapersOnLine* 51, 20 (2018), 381–387.
[127] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*. 1273–1282.
[128] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated learning of deep networks using model averaging. arXiv:1602.05629. Retrieved from http://arxiv.org/abs/1602.05629.
[129] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, et al. 2017. Mixed precision training. arXiv:1710.03740. Retrieved from https://arxiv.org/abs/1710.03740.
[130] Shervin Minaee, Yuri Y. Boykov, Fatih Porikli, Antonio J. Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. 2021. Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), 1–1.
[131] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2013. Memory limited, streaming PCA. In *Advances in Neural Information Processing Systems*. 2886–2894.
[132] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2012. *Foundations of Machine Learning*. MIT Press.
[133] Robert Morris. 1978. Counting large numbers of events in small registers. *Commun. ACM* 21, 10 (1978), 840–842.
[134] Dana Moshkovitz and Michal Moshkovitz. 2017. Mixing implies lower bounds for space bounded learning. In *Conference on Learning Theory*. 1516–1566.
[135] Dana Moshkovitz and Michal Moshkovitz. 2018. Entropy samplers and strong generic lower bounds for space bounded learning. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[136] Michal Moshkovitz and Naftali Tishby. 2017. A general memory-bounded learning algorithm. arXiv:1712.03524. Retrieved from https://arxiv.org/abs/1712.03524.
[137] Tomoya Murata and Taiji Suzuki. 2018. Sample efficient stochastic gradient iterative hard thresholding method for stochastic sparse linear regression with limited attribute observation. In *Advances in Neural Information Processing Systems*. 5317–5326.
[138] M. G. Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. 2019. Machine learning at the network edge: A survey. arXiv:1908.00080. Retrieved from https://arxiv.org/abs/1908.00080.
[139] Apache MXNet. Memory cost of deep nets under different allocations. Retrieved December 22, 2020 https://github.com/apache/incubator-mxnet/tree/master/example/memcost#memory-cost-of-deep-nets-under-different-allocations.

[140] Yurii Nesterov. 2013. *Introductory Lectures on Convex Optimization: A Basic Course*. Vol. 87. Springer Science & Business Media.
[141] Jakob Nielsen. 1993. Usability heuristics. In *Usability Engineering*, Jakob Nielsen (Ed.). Morgan Kaufmann, San Francisco, CA, 115–163. https://doi.org/10.1016/B978-0-08-052029-2.50008-5
[142] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. Paleo: A performance model for deep neural networks. (unpublished).
[143] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing network designspaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10428–10436.
[144] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*. Springer, 525–542.
[145] Sujith Ravi. 2017. Projectionnet: Learning efficient on-device deep networks using neural projections. arXiv:1708.00630. Retrieved from https://arxiv.org/abs/1708.00630.
[146] Ran Raz. 2017. A time-space lower bound for a large class of learning problems. In *Proceedings of the 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS'17)*. IEEE, 732–742.
[147] Baidu Research. Benchmarking deep learning operations on different hardwares. Retrieved December 22, 2020 from https://github.com/baidu-research/DeepBench.
[148] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. 2016. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 18.
[149] Crefeda Faviola Rodrigues, Graham Riley, and Mikel Luján. 2018. Fine-grained energy and performance profiling framework for deep convolutional neural networks. arXiv:1803.11151. Retrieved from https://arxiv.org/abs/1803.11151.
[150] Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. 2016. Delight: Adding energy dimension to deep neural networks. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 112–117.
[151] Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. 2017. TinyDL: Just-in-time deep learning solution for constrained embedded systems. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'17)*. 1–4.
[152] Daniil Ryabko. 2007. Sample complexity for computational classification problems. *Algorithmica* 49, 1 (2007), 69–77.
[153] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.
[154] Rocco A. Servedio. 2000. Computational sample complexity and attribute-efficient learning. *J. Comput. Syst. Sci.* 60, 1 (2000), 161–178.
[155] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
[156] Shai Shalev-Shwartz, Ohad Shamir, and Eran Tromer. 2012. Using more data to speed-up training time. In *Artificial Intelligence and Statistics*. 1019–1027.
[157] Ohad Shamir. 2014. Fundamental limits of online and distributed algorithms for statistical learning and estimation. In *Advances in Neural Information Processing Systems*. 163–171.
[158] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. Retrieved from https://arxiv.org/abs/1409.1556.
[159] Daniel Soudry, Itay Hubara, and Ron Meir. 2014. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*. 963–971.
[160] Nathan Srebro and Karthik Sridharan. 2011. Theoretical basis for "more data less work.". In *Proceedings of the NIPS Workshop on Computataional Trade-offs in Statistical Learning*.
[161] Dimitrios Stamoulis, Ermao Cai, Da-Cheng Juan, and Diana Marculescu. 2018. HyperPower: Power-and memory-constrained hyper-parameter optimization for neural networks. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*. IEEE, 19–24.
[162] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. 2019. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 481–497.
[163] Jacob Steinhardt and John Duchi. 2015. Minimax rates for memory-bounded sparse linear regression. In *Proceedings of the Conference on Learning Theory*. 1564–1587.
[164] Jacob Steinhardt, Gregory Valiant, and Stefan Wager. 2016. Memory, communication, and statistical queries. In *Proceedings of the Conference on Learning Theory*. 1490–1516.

[165] Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. 2018. Sparsified SGD with memory. In *Advances in Neural Information Processing Systems*. 4447–4458.
[166] 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*.
[167] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.
[168] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.
[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*. 2818–2826.
[170] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. 2018. Sketching linear classifiers over data streams. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 757–772.
[171] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2820–2828.
[172] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 6105–6114.
[173] Mingxing Tan and Quoc V. Le. 2019. Mixconv: Mixed depthwise convolutional kernels. In *Proceedings of the British Machine Vision Conference*.
[174] Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. 2018. Communication compression for decentralized training. In *Advances in Neural Information Processing Systems*. 7652–7662.
[175] Zhenheng Tang, Shaohuai Shi, Xiaowen Chu, Wei Wang, and Bo Li. 2020. Communication-efficient distributed deep learning: A somprehensive survey. arXiv:cs.DC/2003.06307. Retrieved from https://arxiv.org/abs/2003.06307.
[176] Konstantinos I. Tsianos, Sean Lawlor, and Michael G. Rabbat. 2012. Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In *Proceedings of the 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton'12)*. IEEE, 1543–1550.
[177] Leslie G. Valiant. 1984. A theory of the learnable. *Commun. ACM* 27, 11 (1984), 1134–1142.
[178] Ewout van den Berg, Bhuvana Ramabhadran, and Michael Picheny. 2017. Training variance and performance evaluation of neural networks in speech. In *Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'17)*. IEEE, 2287–2291.
[179] Jesper E. Van Engelen and Holger H. Hoos. 2020. A survey on semi-supervised learning. *Mach. Learn.* 109, 2 (2020), 373–440.
[180] Vladimir Vapnik. 2006. *Estimation of Dependences Based on Empirical Data*. Springer Science & Business Media.
[181] Vladimir Vapnik and Rauf Izmailov. 2019. Rethinking statistical learning theory: Learning using statistical invariants. *Mach. Learn.* 108, 3 (2019), 381–423.
[182] V. N. Vapnik and A. Ya. Chervonenkis. 1971. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.* 16, 2 (1971), 264–280.
[183] Delia Velasco-Montero, Jorge Fernández-Berni, Ricardo Carmona-Galán, and Angel Rodríguez-Vázquez. 2018. Performance analysis of real-time DNN inference on raspberry Pi. In *Proceedings of the Real-Time Image and Video Processing Conference*, Vol. 10670. SPIE.
[184] Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Deploying deep neural networks in the embedded space. In *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning*.
[185] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. 2020. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12965–12974.
[186] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8612–8620.
[187] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training deep neural networks with 8-bit floating point numbers. In *Advances in Neural Information Processing Systems*. 7685–7694.
[188] Wenlin Wang, Changyou Chen, Wenlin Chen, Piyush Rai, and Lawrence Carin. 2016. Deep distance metric learning with data summarization. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD'16)*.
[189] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Commun. Surv. Tutor.* 22, 2 (2020), 869–904.

[190] Xin Wang, Fisher Yu, Zi-Yi Dou, and Joseph E Gonzalez. 2017. Skipnet: Learning dynamic routing in convolutional networks. arXiv:1711.09485. Retrieved from https://arxiv.org/abs/1711.09485.

[191] Zhihao Wang, Jian Chen, and Steven C. H. Hoi. 2020. Deep learning for image super-resolution: A survey. *IEEE Trans. Pattern Anal. Mach. Intell* (2020), 1–1.

[192] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. 2018. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*. 1299–1309.

[193] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*. 2074–2082.

[194] 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*. 1509–1519.

[195] Simon Wiedemann, Temesgen Mehari, Kevin Kepp, and Wojciech Samek. 2020. Dithered backprop: A sparse and quantized backpropagation algorithm for more efficient deep neural network training. arXiv:2004.04729. Retrieved from https://arxiv.org/abs/2004.04729.

[196] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. 2019. A survey on neural architecture search. arXiv:1905.01392. Retrieved from https://arxiv.org/abs/1905.01392.

[197] Michael M. Wolf. 2018. Mathematical foundations of supervised learning. (unpublished).

[198] Alexander Wong. 2019. NetScore: towards universal metrics for large-scale performance analysis of deep neuralnetworks for practical on-device edge usage. In *International Conference on Image Analysis and Recognition*. Springer, 15–26.

[199] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10734–10742.

[200] Jiaxiang Wu, Weidong Huang, Junzhou Huang, and Tong Zhang. 2018. Error compensated quantized SGD and its applications to large-scale distributed optimization. In *International Conference on Machine Learning*. PMLR, 5325–5333.

[201] Yun Xie and Marwan A Jabri. 1992. Analysis of the effects of quantization in multilayer neural networks using a statistical model. *IEEE Trans. Neural Netw.* 3, 2 (1992), 334–338.

[202] Guandao Yang, Tianyi Zhang, Polina Kirichenko, Junwen Bai, Andrew Gordon Wilson, and Chris De Sa. 2019. SWALP : Stochastic Weight Averaging in Low Precision Training. In *Proceedings of the 36th International Conference on Machine Learning* (Proceedings of Machine Learning Research), Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 7015–7024. http://proceedings.mlr.press/v97/yang19d.html.

[203] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE, 6071–6079.

[204] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 285–300.

[205] Wenzhuo Yang and Huan Xu. 2015. Streaming sparse principal component analysis. In *Proceedings of the International Conference on Machine Learning*. 494–503.

[206] Yukuan Yang, Lei Deng, Shuang Wu, Tianyi Yan, Yuan Xie, and Guoqi Li. 2020. Training high-performance andlarge-scale deep neural networks with full 8-bit integers. *Neural Networks* 125 (2020), 70–82.

[207] Yuanshun Yao, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2017. Complexity vs. performance: Empirical analysis of machine learning as a service. In *Proceedings of the 2017 Internet Measurement Conference*. ACM, 384–397.

[208] Heiga Zen, Yannis Agiomyrgiannakis, Niels Egberts, Fergus Henderson, and Przemysław Szczepaniak. 2016. Fast, Compact, and High Quality LSTM-RNN Based Statistical Parametric Speech Synthesizers for Mobile Devices. In *Proc. Interspeech*. San Francisco, CA, USA.

[209] Chongsheng Zhang, Changchang Liu, Xiangliang Zhang, and George Almpanidis. 2017. An up-to-date comparison of state-of-the-art classification algorithms. *Expert Syst. Appl.* 82 (2017), 128–150.

[210] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. 2016. The zipml framework for training models with end-to-end low precision: The cans, the cannots, and a little bit of deep learning. arXiv:1611.05402. Retrieved from https://arxiv.org/abs/1611.05402.

[211] Kai Zhang, Chuanren Liu, Jie Zhang, Hui Xiong, Eric Xing, and Jieping Ye. 2017. Randomization or condensation?: Linear-cost matrix sketching via cascaded compression sampling. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 615–623.

[212] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional-neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6848–6856.

[213] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1606.06160. Retrieved from https://arxiv.org/abs/1606.06160.

[214] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* 107, 8 (2019), 1738–1762.

[215] Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. 2018. TBD: Benchmarking and analyzing deep neural network training. arXiv:1803.06905. Retrieved from https://arxiv.org/abs/1803.06905.

[216] Xiaojin Jerry Zhu. 2005. *Semi-supervised Learning Literature Survey*. Technical Report. Department of Computer Sciences, University of Wisconsin—Madison.

Received July 2019; revised December 2020; accepted February 2021