Energy-efficient Training of Distributed DNNs in the Mobile-edge-cloud Continuum

Energy-efficient Training of Distributed DNNs in the Mobile-edge-cloud Continuum

Francesco Malandrino
CNR-IEIIT, CNIT

Carla Fabiana Chiasserini
Politecnico di Torino, CNR-IEIIT, CNIT

Giuseppe Di Giacomo
Politecnico di Torino

***Abstract*—We address distributed machine learning in multi-tier (e.g., mobile-edge-cloud) networks where a heterogeneous set of nodes cooperate to perform a learning task. Due to the presence of multiple data sources and computation-capable nodes, a learning controller (e.g., located in the edge) has to make decisions about (i) which distributed ML model structure to select, (ii) which data should be used for the ML model training, and (iii) which resources should be allocated to it. Since these decisions deeply influence one another, they should be made jointly. In this paper, we envision a new approach to distributed learning in multi-tier networks, which aims at maximizing ML efficiency. To this end, we propose a solution concept, called RightTrain, that achieves energy-efficient ML model training, while fulfilling learning time and quality requirements. RightTrain makes high-quality decisions in polynomial time. Further, our performance evaluation shows that RightTrain closely matches the optimum and outperforms the state of the art by over 50%.**

I. Introduction

The combination of machine learning (ML) and 5G networks enables the so-called Internet of Intelligent Things, whereby user equipments (UEs), such as smartphones and smart-city devices, can leverage cloud-based ML services.

The diversity of the devices performing ML tasks will be matched by the diversity of the tasks themselves. Different applications and scenarios call for different ML approaches, including supervised [1] and unsupervised [2] learning, as well as hybrid approaches. A large number of these tasks can be accomplished through deep neural networks (DNNs), built by combining a sequence of *layers* of different types [1].

However, the performance of distributed learning depends upon three major factors, namely, (i) the quantity of the data used for learning, (ii) the learning strategy itself, e.g., the distributed DNN structure employed, and (iii) the computational and network resources allocated to the learning task. In this paper, we address such decisions by presenting a framework, named *RightTrain*, that makes joint high-quality, energy-efficient decisions about data usage, distributed DNN structure selection, DNN layer-to-physical node mapping, and resource allocation. Our framework can capture the nontrivial (and, often, counter-intuitive) ways in which such choices interact with each other, and yields decisions that are provably close to the optimum, while keeping a low complexity.

Our work is related to studies on distributed ML in edge scenarios. However, while those works [3]–[5] aim at finding the right resources to implement a given and immutable learning task, we take a more flexible approach, and address the important challenge of *adapting* the DNN structure and the resources it uses to one another, thus achieving unparalleled learning efficiency. Besides proposing such new solution concept, we also identify new research directions for distributed ML in multi-tier networks.

II. System Model and Problem Formulation

The system model describes a DNN training task leveraging distributed learning and exploiting the resources of the nodes in the multi-tier network. Such nodes communicate with, and are coordinated by, a *central controller*, typically running at the edge of the network infrastructure, which collects information on the nodes' capabilities and position [3].

Input information. DNN learning tasks are performed through an architecture composed of *layers* of different types (e.g., fully-connected or convolutional). Each layer has a local set of *parameters* defining its behavior (e.g., the weights of a fully-connected layer), and *training* a DNN means finding the parameter values that minimize a global loss function.

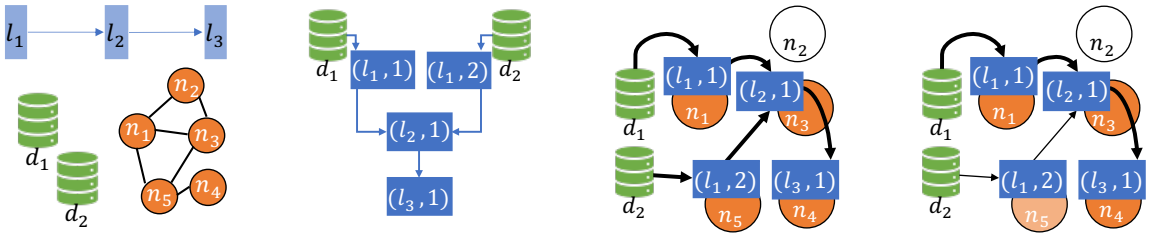
Thus, the input to our problem includes (see Fig. 1(a)):

- a set $\mathcal{L} = \{l_1, \ldots, l_L\}$ of *layers* of the DNN architecture to perform the learning task at hand;
- a set $\mathcal{N}$ of mobile, edge or cloud *physical nodes* with the capability to run one or more DNN layers;
- a set $\mathcal{D}$ of *data sources*.

For each layer $l \in \mathcal{L}$, we indicate with $r(l)$ the computational requirement, which is known and expresses (e.g., in CPU cycles/Mbit) the amount of computing resources required to process one unit of traffic entering layer l, including both forward and backward passes. We are also given coefficients $q(l)$, indicating the ratio between outgoing and incoming data for layer l. Importantly, set $\mathcal{L}$ of DNN layers is given as an input to our problem, as it comes from domain- and application-specific knowledge. *Adapting* the DNN to the available network resources is, instead, one of the decisions we make, as set forth next.

For each node $n \in \mathcal{N}$, we are given the total amount $R(n)$ of available computational resources therein, that can be allocated to the layers running at n. Parameters $\mu(l, n) \in \{0, 1\}$ express whether node n has enough memory to execute layer l.

Concerning data transmission, for any two nodes n_1, n_2, $S(n_1, n_2)$ indicates the amount of data that can be transferred in a time unit over the link (assumed to be symmetric) between the two nodes. Also, $S(n_1, n_2) = \infty$ if $n_1 = n_2$, and $S(n_1, n_2) = 0$ if n_1 and n_2 cannot communicate. Finally, let $\Delta(d)$ be the data generated by source $d \in \mathcal{D}$ at each epoch.



(a) (b) (c) (d)

Fig. 1. (a) Model elements: *layers* of the DNN architecture (l_j, light blue), *data sources* (d_j, green), and *physical* nodes (n_j, orange); (b) a possible distributed DNN structure whose nodes are data sources and layer instances; (c) layer instance-to-node mapping; (d) a resource allocation, using only some of d_2's data and, accordingly, reducing the computing resources allocated to instance $(l_1, 2)$ as underlined by the shade of n_5.

Decision variables. Given the DNN architecture at hand, several *distributed structures* thereof can be envisioned, including data sources and one or more instances of each layer (see Fig. 1(b)); we refer to each of such structures as an *instance tree*. To run an instance tree, we need to identify the physical nodes to use, to allocate resources therein, and to determine the quantity of data to be used for training.

Building the instance tree. For each layer $l \in \mathcal{L}$, at least one and at most $\alpha|\mathcal{D}|$ *layer instances* shall be created, with $\alpha \geq 1$. Each layer instance runs at a physical node, and is identified as a pair (l, i), where $i = 1, \dots \alpha|\mathcal{D}|$; the set of layer instances is denoted by $\mathcal{I}$. As mentioned, layer instances and data sources in $\mathcal{D}$ are connected to form an instance tree, with binary variables $y(l, i, m, j) \in \{0, 1\}$ expressing whether layer instance (l, i) shall be connected to layer instance (m, j). As depicted in Fig. 1(b), each data source $d \in \mathcal{D}$ can be associated with at most one instance $(d, 1)$, and not associating a source with any layer instance means not using it.

Instance-to-node mapping and resource allocation. Given the instance tree and the network topology, the coordinator has to decide which node $n \in \mathcal{N}$ runs layer instance $(l, i) \in \mathcal{I}$; such a decision is expressed through binary variables $z(l, i, n) \in \{0, 1\}$. Also, $\nu(l, i)$ denotes the node running instance (l, i), i.e., such that $z(l, i, \nu(l, i)) = 1$. Another decision concerns computational resources $\rho(l, i) \leq R(\nu(l, i))$ (in CPU cycles/s), to be assigned to each layer instance (l, i).

Training data and traffic. For each data source d, the coordinator determines the quantity $x(d, 1, m, j)$ of data to be transferred towards layer instance (m, j). Given the x-variables, we define $\chi(l, i, m, j)$ as the quantity of data flowing over the link from instance (or source) (l, i) to instance (m, j).

Constraints and objective function. First, the instance tree topology determined by the y-variables must include at least one instance of each layer, i.e., $\sum_{i \in [1 \dots \alpha|\mathcal{D}|]} \sum_{(m,j) \in \mathcal{I}} y(l, i, m, j) \geq 1, \forall l \in \mathcal{L}$, and only subsequent layers can be connected, i.e., $y(l, i, m, j) \leq \mathbb{1}_{[l \text{ is child of } m]}$ With regard to instance-to-node mapping (i.e., z-variables), each layer instance must be mapped onto exactly one node, i.e., $\sum_{n \in \mathcal{N}} z(l, i, n) = 1, \forall (l, i) \in \mathcal{I}$ and no layer instance can be mapped onto a node lacking the required memory resources, i.e., $z(l, i, n) \leq \mu(l, n), \forall (l, i) \in \mathcal{I}, n \in \mathcal{N}$.

As for the ρ- and x-variables, the available computing and communication resources cannot be exceeded, i.e.,

$$\sum_{(l,i) \in \mathcal{I}:\ \nu(l,i)=n} \rho(l, i) \leq R(n), \quad \forall n \in \mathcal{N}, \tag{1}$$

$$\sum_{\substack{(l,i):\ \nu(l,i)=n \\ (m,j):\ \nu(m,j)=n'}} \chi(l, i, m, j) \leq y(l, i, m, j) S(n, n'), \quad \forall n, n' \in \mathcal{N}. \tag{2}$$

Finally, the data fed by a data source into a layer instance cannot exceed $\Delta(d)$, i.e., $x(d, 1, l, i) \leq \Delta(d), \forall d, (l, i)$.

Decisions x, y, z and ρ fully describe the behavior of the DNN training task. However, they do not directly express: (i) the time taken by each learning epoch; (ii) the energy consumed in each learning epoch; (iii) the number of epochs needed to attain the required learning quality, i.e., the maximum allowable value of loss function $\epsilon^{\max}$. We account for these quantities through functions $\mathbf{T}(x, y, z, \rho)$, $\mathbf{E}(x, y, z, \rho)$, and $\mathbf{K}(y, x, \epsilon)$, respectively. Note that $\mathbf{T}(x, y, z, \rho)$ and $\mathbf{E}(x, y, z, \rho)$ are straightforward to characterize using [6], linking the CPU and memory resources associated with each layer to the corresponding energy consumption. Furthermore, $\mathbf{E}(x, y, z, \rho)$ values can be normalized to account for the fact that saving energy is more critical for some devices, e.g., battery-powered ones, than for others. Finally, it is important to highlight how $\mathbf{T}(x, y, z, \rho)$ captures the transfer times between devices as well as the computation times. $\mathbf{K}(y, x, \epsilon)$ can instead be obtained through different means, e.g., testbed measurements [7], which also include the effect of the quantity and quality of available data on accuracy. All such functions represent inputs to our problem.

Given $\mathbf{T}$, $\mathbf{E}$, and $\mathbf{K}$, we formulate our problem of minimizing the learning energy consumption, subject to the constraints above and to achieving loss $\epsilon^{\max}$ by time $T^{\max}$, as:

$$\min_{x,y,z,\rho} \mathbf{K}(y, x, \epsilon^{\max}) \mathbf{E}(x, y, z, \rho) \tag{3}$$

$$\text{s.t.} \mathbf{K}(y, x, \epsilon^{\max}) \mathbf{T}(x, y, z, \rho) \leq T^{\max}. \tag{4}$$

III. The RightTrain Solution

Directly optimizing (3) subject to constraints (4) is a daunting task. We thus introduce the RightTrain solution concept, whose main steps are summarized in Fig. 2. It takes as an input the set of instance trees to consider (like the one in Fig. 1(b)); such a set can be efficiently computed

offline, in a scenario- and application-dependent manner. Then, *temporarily* assuming that all data and resources will be used, RightTrain iterates over the instance trees, selecting at each step the one requiring the least amount of *total* processing (Step 1 in Fig. 2, detailed in Sec. III-A). For each tree, the y-variables are given, hence, in Step 2 (Sec. III-B) RightTrain makes the mapping decisions z, still keeping the *temporary* assumption that all data and resources will be used. This assumption is dropped in Step 3 (Sec. III-C), which seeks to refine the solution obtained in Step 2 by using less data and/or less computing power, thereby reducing the energy consumption without jeopardizing the learning performance. If a feasible solution is obtained, then the algorithm terminates (Step 4); otherwise, it goes back to Step 1 and moves to the next instance tree.

A. Instance tree ordering

Ideally, in Step 1 of the RightTrain solution one would like to select a tree minimizing the energy consumption (3); however, this is not possible as instance-to-node mapping and resource assignment decisions (respectively, Steps 2–3) have yet to be made. Nonetheless, the instance tree – along with information on layer and data source characteristics – allows estimating the total quantity of *processing* entailed by the tree itself, which can be expressed as:

$$\mathbf{K}(y, \Delta, \epsilon^{\max}) \sum_{l:\ (l,i)\in\mathcal{I}} r(l) \sum_{\substack{d\in\mathcal{D}:\ (l,i) \text{ is} \\ \text{an ancestor of } d}} \Delta(d) \prod_{\substack{m\in\mathcal{L} \text{ in path} \\ \text{from } d \text{ to } l}} q(m). \quad (5)$$

Looking at (5) from right to left, the processing required by a given layer l of a DNN for each epoch depends upon [8]: (i) the incoming data (which in turn depends upon the q-coefficients of the layers traversed before l) and (ii) the layer complexity. Such quantities are then summed across all layer instances, and multiplied by the number of epochs to run.

B. Layer instance-to-node mapping

Step 3 of RightTrain *maps* layer instances in $\mathcal{I}$ onto nodes in $\mathcal{N}$. As mentioned, initially such decisions are made assuming that all available data and computational capabilities are used. The resulting problem is combinatorial and, in general, hard to approach; on the positive side, however, we can leverage the *tree* structure connecting layer instances, as exemplified in Fig. 1(b). We do so by:
(i) building the *expanded graph*, summarizing all viable mapping decisions, and

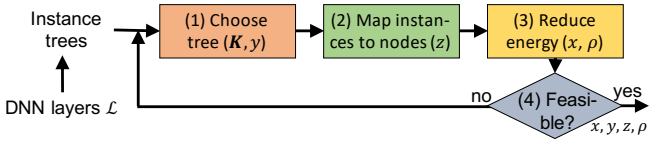


Fig. 2. Main steps of RightTrain: given the DNN architecture at hand, at each iteration RightTrain selects the DNN structure (i.e., instance tree) with the lowest computational load (Step 1, Sec. III-A). It then makes the near-optimal layer instance-to-node mapping (Step 2, Sec. III-B), and further improves efficiency by tweaking data and resource utilization (Step 3, Sec. III-C).

Algorithm 1 Step 3 of RightTrain

Require: Expanded graph $\{d\} \cup \{(l,i,n)\} \cup \{\Omega\}$
1: $\mathcal{T} \leftarrow \{\Omega\}$
2: **while** $\mathcal{D} \setminus \mathcal{T} \not\equiv \emptyset$ **do**
3: $\quad w^\star, \pi^\star \leftarrow \infty, \emptyset$
4: $\quad$ **for all** $d \in \mathcal{D} \setminus \mathcal{T}, v \in \mathcal{T}$ **do**
5: $\quad\quad w, \pi \leftarrow \mathsf{RestrictedMinWeightPath}(d, v, T^{\max})$
6: $\quad\quad$ **if** $w < w^\star$ **then**
7: $\quad\quad\quad w^\star, \pi^\star \leftarrow w, \pi$
8: $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \pi^\star$
9: **for all** $(l,i,n) \in \mathcal{T}$ **do**
10: $\quad z(l,i,n) \leftarrow 1$

(ii) computing a delay-aware Steiner tree (DA-ST) on the expanded graph, i.e., the minimum-weight tree spanning the vertices that correspond to the considered instance tree, with the additional constraint on the maximum learning time. By construction, the topology of the DA-ST corresponds to near-optimal mapping decisions.

The vertices of the DA-ST tree represent the layer instance-to-node mapping minimizing energy consumption (3), under the above conditions, i.e., that all data and resources are used. The mapping between vertices and decisions is made as summarized in Alg. 1. Given the expanded graph, in Line 1 we initialize the DA-ST $\mathcal{T}$, so as to include vertex Ω. Then, so long as there are data sources not yet included in the tree (Line 2), we look for the minimum-weight path connecting a data source d not yet in $\mathcal{T}$ with a vertex v in $\mathcal{T}$, subject to (4). To this end, we leverage function $\mathsf{RestrictedMinWeightPath}(d, v, T^{\max})$ (Line 5), providing the path π connecting each data source d not yet reached by $\mathcal{T}$ with each vertex v already in the tree, as per [9]. The minimum-weight path is then identified (Line 7) and added to the DA-ST $\mathcal{T}$ in Line 8. Once all data sources have been included, $\mathcal{T}$ is complete. The algorithm therefore sets to 1 the z-variables corresponding to the selected DA-ST vertices (Line 10), and returns them.

In addition to providing high-quality decisions, Alg. 1 has a remarkably low (namely, quadratic) computational complexity, as can be seen by inspecting the algorithm itself. Indeed, the outermost loop is performed at most $|\mathcal{D}|$ times, and the inner one at most $|\mathcal{T}|$ times.

C. Optimizing data and resource usage

In Steps 2 and 3, we have set the y-variables and z-variables. The values of both variable sets have been obtained under the assumption that all data from the selected data sources and all the capabilities of the involved physical nodes are used. Next, Step 3 seeks to establish whether *all* that data and that computational power are really needed. Our goal is thus to obtain a solution that meets all constraints in (4), while further improving the energy objective in (3).

Given the y- and z-values, the problem in (3)–(4) only has continuous variables, i.e., x and ρ. Such continuous problems can be efficiently tackled by off-the-shelf solvers or through iterative gradient-based methods like BFGS, with polynomial worst-case complexity [10], and much faster in most cases.

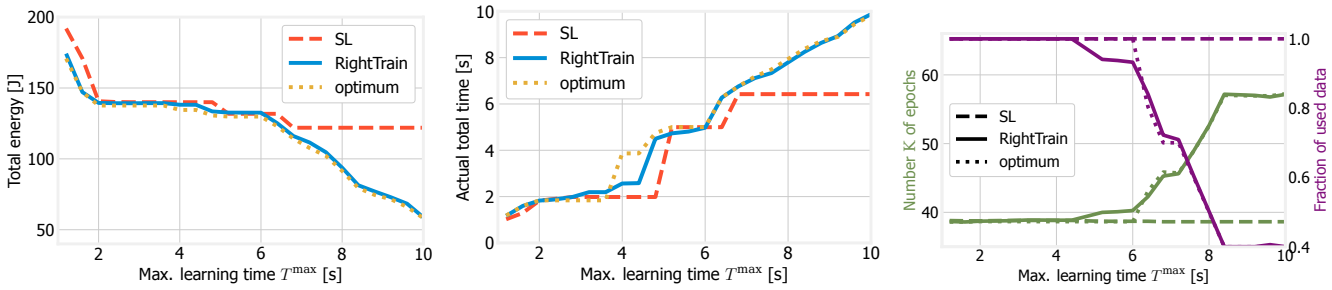


Fig. 3. Energy consumption (left), actual (center), number of epochs and fraction of used data (right), vs. maximum learning time.

IV. Performance Evaluation

We now seek to quantify how RightTrain improves the performance of a simple, yet representative, image classification task, based on the widely popular AlexNet [11] DNN and the *de facto* standard CIFAR dataset. We consider mobile-edge-cloud scenario including 4 data sources and 5 mobile nodes.

We compare the performance of RightTrain against split learning (SL) [4], owing to the relevance and good performance of the latter [12]. In the considered scenarios, SL splits the DNN into three parts, and aims at running one in each of the mobile, edge, and cloud network segments. Since we are interested in the best decisions that can be made under the SL paradigm, we try out all possible splits, and choose the one resulting in the best value of the objective in (3). We also compare against the optimum, i.e., the decisions that minimize the total energy consumption, as obtained via brute force.

We compare the behavior of RightTrain and the alternatives approach with respect to the most relevant performance metrics, namely, energy consumption and learning time. Fig. 3 (left) shows the energy consumed versus the maximum learning time $T^{\max}$. As expected, lower values of $T^{\max}$, hence, tighter delay constraints, result in higher energy consumption. Also, we can identify two distinct operational regions. When $T^{\max}$ is small, all strategies perform similarly, with RightTrain consuming slightly less energy than SL and close to the optimum, owing to its greater flexibility in layer instance-to-node mapping. As $T^{\max}$ grows, the energy consumed by SL remains unchanged, while RightTrain matches the optimum and consumes over 50% less energy than SL. The reason is shown in Fig. 3 (center): SL yields shorter learning times than the maximum, especially for high $T^{\max}$.

This is confirmed by the results in Fig. 3 (right), portraying the fraction of data used by each strategy (purple) and the resulting number of epochs $\mathbf{K}$ (green). SL (dashed lines) always uses all available data, which results in a constant (and low) number of epochs. Conversely, both RightTrain and the optimum can use less data when the delay requirements are looser, achieving a lower energy consumption and, hence, a better efficiency, in spite of a higher number of epochs.

V. Conclusion and Future Work

We have presented RightTrain, an effective scheme to achieve energy-efficient ML training in the mobile-edge-cloud continuum. RightTrain is predicated on making joint decisions on (i) selecting the data to be used, (ii) choosing the distributed DNN structure, and (iii) allocating the resources to train it. Owing to its holistic nature, RightTrain is able to adapt the learning tasks to the available computational resources, networking capabilities, and data; this is in contrast with state-of-the-art works, which consider the learning task given and seek to find the resources to support them.

Future work should further investigate the trade-off between learning performance and energy efficiency, and find new paradigms to leverage at best the distributed resources offered by a multi-tier network. Additionally, new solutions are needed to optimally cope with the randomness affecting the topology and operational conditions of mobile nodes that get involved in the learning process.

Acknowledgement

This publication was made possible by NPRP-S 13th Cycle grant no. NPRP13S-0205-200265 from the Qatar National Research Fund (a member of Qatar Foundation). The findings herein reflect the work, and are solely the responsibility, of the authors.

References

[1] C. Zhuang, A. L. Zhai, and D. Yamins, "Local aggregation for unsupervised learning of visual embeddings," in *IEEE/CVF ICCV*, 2019.
[2] M. Mancini *et al.*, "Inferring latent domains for unsupervised deep domain adaptation," *IEEE TPAMI*, 2019.
[3] E. Li *et al.*, "Edge AI: on-demand accelerating deep neural network inference via edge computing," *IEEE Trans. on Wireless Comm.*, 2019.
[4] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.
[5] S. Deng *et al.*, "Edge Intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, 2020.
[6] E. García-Martín *et al.*, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, 2019.
[7] Y. E. Wang *et al.*, "Benchmarking tpu, gpu, and cpu platforms for deep learning," *arXiv preprint arXiv:1907.10701*, 2019.
[8] R. Schwartz *et al.*, "Green AI," *Comm. ACM*, 2020.
[9] D. H. Lorenz *et al.*, "A simple efficient approximation scheme for the restricted shortest path problem," *Operations Research Letters*, 2001.
[10] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*, 2004.
[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, 2012.
[12] Y. Gao *et al.*, "End-to-end evaluation of federated learning and split learning for internet of things," in *IEEE SRDS*, 2020.