

Distributed Training of Graph Convolutional Networks

Simone Scardapane, Indro Spinelli, *Student Member, IEEE*, and Paolo Di Lorenzo, *Senior Member, IEEE*

Abstract—The aim of this work is to develop a fully-distributed algorithmic framework for training graph convolutional networks (GCNs). The proposed method is able to exploit the meaningful relational structure of the input data, which are collected by a set of agents that communicate over a sparse network topology. After formulating the centralized GCN training problem, we first show how to make inference in a distributed scenario where the underlying data graph is split among different agents. Then, we propose a distributed gradient descent procedure to solve the GCN training problem. The resulting model distributes computation along three lines: during inference, during back-propagation, and during optimization. Convergence to stationary solutions of the GCN training problem is also established under mild conditions. Finally, we propose an optimization criterion to design the communication topology between agents in order to match with the graph describing data relationships. A wide set of numerical results validate our proposal. To the best of our knowledge, this is the first work combining graph convolutional neural networks with distributed optimization.

Index Terms—Graph convolutional networks, distributed optimization, networks, consensus.

I. INTRODUCTION

Nowadays, graph-based data are pervasive, with applications ranging from social networks [1] to epidemiology [2], recommender systems [3], cyber-security [4], sensor networks [5], natural language processing [6], genomics [7], and many more. In fact, graphs are the natural format for any data where the available structure is given by pairwise relationships.

In the field of machine learning, graphs can be found in two varieties. In the case of *graph models*, such as graph neural networks (GNNs) [5], [8]–[10], graph kernels [11], [12], or manifold regularization techniques [13], [14], input data are provided in the form of a graph (such as relations among friends in a social network), and the model must exploit this relational structure when evaluating its output. For example, many classes of GNNs alternate local operations on the components of the graph with message-passing steps between neighboring nodes [10]. In this case, edges represent *semantic* connections between two data points, such as relations in a knowledge base or friendship in a social network.

Alternatively, in *distributed optimization* of machine learning models, the graph represents the underlying physical communication pattern between agents, which possess only a partial view on the overall dataset and/or model. For convex models, notable examples include the alternating direction

method of multipliers (ADMM) [15], [16], diffusion optimization [17], and distributed sub-gradient methods [18]. More recently, several distributed optimization models have been proposed also in the non-convex case [19]–[23], with applications to machine learning [24], [25]. In this setting, the graph imposes stringent constraints on the way information can be propagated across the nodes, i.e., an operation is allowed only if its implementation can be done through message passing between two neighboring agents.

The majority of works on distributed optimization of machine learning models assumes that different examples (be they vectors, images, or similar) are independently distributed across physical agents, with no relation among them. In several situations, however, data has meaningful relational structure that can be exploited. In this context, a distributed optimization algorithm should be able to exploit both kinds of connections, physical and semantic. In this paper, we investigate a problem at the boundary of these two worlds. In particular, we assume that a graph is available representing similarity among data points, but these points are also physically distributed among multiple agents. A second *communication* graph then represents feasible communication paths among the agents.

Our algorithm is tailored to situations in which data is naturally represented in terms of a graph, requiring graph neural network models for its processing. Furthermore, data are collected by a set of independent agents, and sharing local information with a central processor is either unfeasible or not economical/efficient, owing to the large size of the network and volume of data, time-varying network topology, energy constraints, and/or privacy issues. For example, this is the case of sensor networks [26], in which energy requirements restrict communication between sensors, and at the same time their spatial relation correlates their measurements. A practical case is studied in Sec. V-F, where we consider a network of distributed traffic sensors. We assume that the sensors send data to a fixed set of base stations (the *agents* in our paper), which in turn communicate between them to perform the training of a graph neural network model to forecast traffic. Similarly, we can think of medical entities possessing text databases, wherein distributed communication is used to ensure privacy, while the data graph can describe relations (e.g., citations) among texts and/or patients.

Related works

In this paragraph, we review the relevant literature on graph-based machine learning, optimization, and signal processing, considering separately the single research areas.

Distributed optimization: Classical machine learning techniques assume that the training data is given by a set of

Corresponding author email: simone.scardapane@uniroma1.it

The authors are with the Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Italy. Email: {simone.scardapane,indro.spinelli,pao.lorenzo}. This work was funded by “Bandi di Ateneo per la Ricerca 2019”, Sapienza University of Rome.

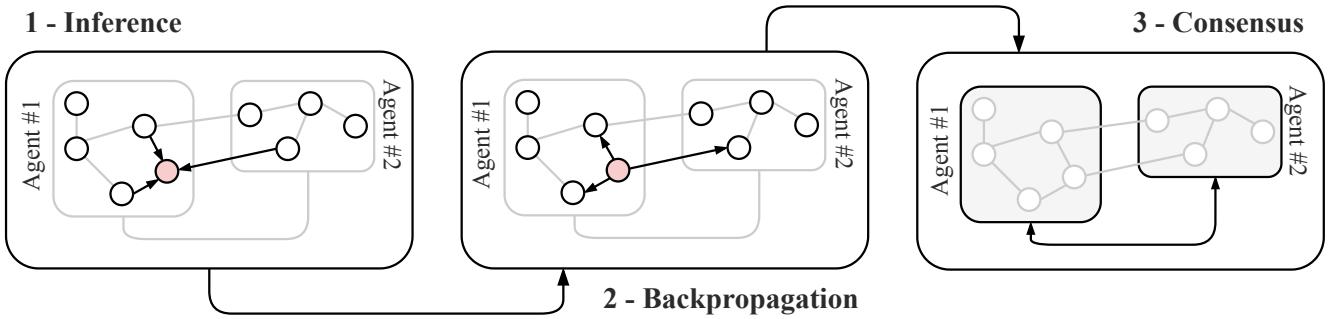


Fig. 1. Illustration of the proposed approach. In step 1, nodes communicate to perform inference. In step 2, a symmetric communication phase is executed to compute local gradients. In step 3, agents exchange local variables to asymptotically reach agreement. For steps 1-2, a representative active node is shown in red. Directed arrows show the flow of messages.

independently distributed samples. In distributed optimization, it is further assumed that these samples are distributed across multiple agents having only sparse connectivity, either because of communication or privacy concerns [27]. The objective is to devise globally convergent algorithms that can be implemented with local exchanges across agents. As mentioned in the introduction, this class of approaches includes ADMM [15], diffusion methods [17], non-convex distributed gradient descent [19], sub-gradient methods [18], [28], projection-based methods [29]–[31], and several others, e.g., [20]–[22]. Some works can also be found in the specific application of distributed optimization of neural networks [24]. Differently from this class of works, in this paper we assume that the data has an additional relational structure, in the form of pairwise connections between training points. Such structure typically induces a coupling among the objective functions of the agents, which must be properly taken into account in the development of tailored distributed optimization methods. In fact, graph data has been receiving renewed interest in the machine learning literature, as detailed next.

Graph signal processing: The field of graph signal processing (GSP) is motivated with extending classical signal processing tools to graph-based data [32]. Notable results include the definition of a graph Fourier transform [32], uncertainty principle [33], graph filters [34]–[36], and adaptive reconstruction strategies [37]. Closer to our work, several authors have considered completely distributed strategies for GSP techniques, including distributed recursive least-squares [37], [38], distributed graph filtering [39], diffusion adaptation for graph signals [40], [41], and also kernel-based approaches [5]. Linear and kernel methods, however, lack the flexibility in modelling nonlinear data with respect to modern deep neural networks. In addition, the majority of these works assume that the connections between data points exactly match the communication structure underlying the agents; an assumption that we relax in this paper.

Graph neural networks: Similarly to GSP, the field of graph neural networks (GNNs) tries to extend the impressive results of deep neural networks to graph-based data. The literature on GNNs is vast and we refer to recent surveys for an overview [10], [42]–[44]. Most GNNs can be categorized in three broad classes: message-passing neural networks (MPNNs) [45], ex-

tensions of recurrent networks to graph-based data [46], and non-linear extensions of GSP techniques, going under the broad name of geometric deep learning [10]. The last class originates from [8], which proposed to use graph Fourier transforms to replicate classical filtering operations used in convolutional neural networks (CNNs). Depending on the type of graph filter used for the graph convolution, a number of GNNs with varying characteristics can be obtained [9], [47], [48]. In the limiting case of a linear filter, the resulting network only requires messages with 1-hop neighbors and can be implemented as a MPNN. The resulting order-1 GCN [9] (or simply GCN for shortness in the paper) has gained widespread popularity, especially for semi-supervised learning on graphs. While the output of a GNN (and the corresponding back-propagation rule) naturally decomposes over the nodes of the graph [46], its computation requires that all nodes possess a shared copy of the GNN weights, which is not feasible in the distributed scenario considered here.

Parallel and federated learning: Several authors have considered *parallel* implementations of GNNs [49], [50] to exploit multiple workers in parallel to optimize large-scale GCNs. The aim of these works, however, is only to scale and speed up training, with the possibility of freely exchanging data and parameters between the coordinator and the workers. More in general, federated learning (FL) [51], [52] considers the scenario where data belongs to multiple owners, with possibly strong privacy and/or communication constraints. The vast majority of FL works, however, considers the presence of one or more coordinators of the training process, while we work in a fully-distributed scenario. In addition, no author to the best of our knowledge has considered a scenario where both data and agents have a meaningful connectivity structure.

Contributions of the paper

In this paper, we propose the first algorithm for distributed training of GCNs in the scenario described above. We extend the basic graph convolutional network (GCN) model [9], incorporating ideas from distributed optimization [29]–[31] to perform inference and training using only message-passing exchanges among different agents. Interestingly, the resulting model is naturally distributed along three different lines: (i) during prediction, agents communicate among them according

to the standard GCN model to compute their output; (ii) during backpropagation, a symmetric set of communication steps is executed to compute a local gradient for updating the model agent-wise; and (iii) a local consensus step is done to ensure convergence to a common architecture across the entire network. We refer to Fig. 1 for a pictorial description of the approach. The convergence properties of the proposed strategy to stationary solutions of the GCN training problem are also analyzed in some detail. The method hinges on distributed projected gradient methods [29]–[31], extending them to the case of nonseparable objective functions, whose coupling comes from the relational structure given by the data graph.

In our formulation, we assume that the data graph is given, while the user has a certain flexibility in designing the connectivity between the physical agents. To this end, we also propose a simple (convex) criterion to optimize the connectivity between agents and ensure convergence of the distributed GCN training, given the underlying constraints arising from the data graph. The convex problem is solved using a low-complexity ADMM algorithm, which is applicable also in the case of large number of network agents.

We experimentally evaluate our distributed GCN on a number of benchmark datasets for graph neural network models, associating a data sub-graph to each different agent. We illustrate that our protocol can reach performance on par with a standard centralized GCN, and vastly superior to any neural network that does not exploit the relational structure in the data. Also, we evaluate the robustness of the model in the case where the communication graph between agents is not consistent with the structure of the data, and we show that it scales favorably even for sparse communication graphs. Finally, we apply our method to a vehicular traffic prediction problem, using real data collected by a network of sensors located in the San Francisco area [53].

Organization of the paper: In Sec. II we introduce the basic GCN model. Sec. III, the main innovative contribution of the work, describes our distributed setup, the proposed training algorithm, and its convergence properties. In Sec. IV, we describe a criterion to optimize the agents’ connectivity in order to match with the data graph. Finally, we numerically validate the proposed approach in Sec. V, before drawing some conclusions in Sec. VI.

A note on indexing: Throughout the paper, we consider two types of graphs, either related to the data (*data graph*) or related to the connectivity of the physical agents (*communication graph*). To simplify reading, we use i, j, l to index nodes in the data graph, whereas k and z are used to index agents inside the communication graph.

II. GRAPH CONVOLUTIONAL NETWORKS

In this section, we introduce the GCN model [43]. We focus on order-1 GCNs [9] (corresponding to an order-1 FIR filter over the graph [48]) to enhance readability. Our framework can however be applied to any graph neural network that can be implemented via iterative message-passing over a graph, e.g., GAT [54], GINN [55], and several others. In particular, we

show an extension to order- P GCNs, equivalent to a generic FIR filter over the graph [48], [56], in Section III-E.

Consider a generic *data graph* $\mathcal{G}_D = (\mathcal{V}_D, \mathcal{E}_D)$, where $\mathcal{V}_D = \{1, \dots, n\}$ is the set of n vertexes (each corresponding to a data point), and \mathcal{E}_D a set of edges between them. The edges may encode different kinds of relationships among the data as, e.g., similarities, correlations, causal dependencies, etc. The graph is equivalently described by a (weighted) matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ encoding adjacency information (i.e., a shift operator, in the GSP terminology). Every vertex is associated with a data vector $\mathbf{x}_i \in \mathbb{R}^d$ (e.g., a bag-of-words for a text, or a vector of features for a user in a social network). A subset $\mathcal{T}_D \subset \mathcal{V}_D$ of vertexes has associated a corresponding label y_i , $i \in \mathcal{T}_D$. The task of (semi-supervised) node classification aims at assigning a label also to the remaining set $\mathcal{V}_D \setminus \mathcal{T}_D$.

An order-1 graph convolutional (GC) layer is defined by (ignoring biases for simplicity, but w.l.o.g.):

$$\mathbf{H} = \phi(\mathbf{DXW}), \quad (1)$$

where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is a matrix collecting all vertex features row-wise, $\mathbf{W} \in \mathbb{R}^{d \times q}$ is a matrix of trainable coefficients, and ϕ is an element-wise non-linear function (e.g., for ReLU $\phi(s) = \max(0, s)$). The matrix \mathbf{D} encodes edge information in the data graph, and can be chosen as any graph shift operator such as the Laplacian matrix, or normalized versions of it.

A graph convolutional network (GCN) f is built by stacking multiple layers in the form (1). For example, a two-layered GCN for classification is described by:

$$f(\mathbf{W}, \mathbf{V}; \mathbf{X}) = \text{softmax}(\mathbf{D}\phi(\mathbf{DXW})\mathbf{V}), \quad (2)$$

where \mathbf{W} and \mathbf{V} are the two matrices of adaptable coefficients. The generalization to more than two layers is straightforward. Note that the GCN in (2) acts on the data graph through the matrix \mathbf{D} . For instance, the output of (2) for a single data point depends on the 2-hop neighbourhood of the node. In general, for a GCN with K layers of the form (1), the output for node i will depend on neighbours up to K steps away. We denote this set by \mathcal{N}_i^K . Also, denote by \hat{y}_i the prediction of the GCN on node i (i.e., the i -th element of f), and by \mathbf{w} the vector collecting all the trainable coefficients of f [e.g., $\mathbf{w} = [\text{vec}(\mathbf{W}); \text{vec}(\mathbf{V})]$ in (2)]. Then, the optimal trained vector \mathbf{w}^* of the GCN is obtained by solving the optimization problem:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \left\{ \frac{1}{|\mathcal{T}_D|} \sum_{i \in \mathcal{T}_D} l(\mathbf{w}; \hat{y}_i, y_i) \right\}, \quad (3)$$

where $l(\mathbf{w}; \cdot, \cdot)$ is a suitable loss function, e.g., squared loss for prediction, or cross-entropy for multi-class classification. Problem (3) is generally solved in a centralized manner by gradient descent or some accelerated first-order algorithm [43].

III. DISTRIBUTED GRAPH CONVOLUTIONAL NETWORKS

In this section, we introduce our model for distributed GCNs, devising a simple, but very effective, algorithmic solution that performs inference and training using only message-passing exchanges in the neighbourhood of each agent.

A. Problem setup

Let us consider a setting in which the vertexes of the data graph \mathcal{V}_D (i.e., the data points) are distributed across a set of physically separated agents (Fig. 1, left, where every white node represents a vertex in the data graph \mathcal{V}_D). We assume that the number m of agents is much lower than the number n of data points, i.e., $m \ll n$. The agents are physically distributed, and they can communicate among them according to a *communication graph* (or agents' graph) $\mathcal{G}_C = (\mathcal{V}_C, \mathcal{E}_C)$. That is, agent k can send messages to agent z only if the link $(k, z) \in \mathcal{E}_C$.¹ No centralized coordinator is assumed to be available. Node i in the data graph is assigned to agent $a(i) \in \mathcal{V}_C$, where we refer to $a(\cdot)$ as the assignment function that matches data points with agents. The data graph and assignment function are supposed to be given in the current problem setup. We also assume that, whenever there is an edge $(i, j) \in \mathcal{E}_D$ in the data graph, there is a corresponding viable communication path $(a(i), a(j)) \in \mathcal{E}_C$.² Once this assumption is satisfied, the rest of the communication graph can be optimized to balance the amount of messages exchanged on the agents' network and the speed of convergence of our algorithm. In Section IV, we describe an optimization procedure to select a connectivity matrix that satisfies both the agreement of the agents' graph with the data graph, and a user-selected level of global connectivity.

Before proceeding, we introduce some additional notation that will be helpful in the sequel. The set of training points available to agent k is given by:

$$\mathcal{T}_k = \{i \mid a(i) = k\}, \quad (4)$$

that is, training points assigned by $a(i)$ to agent k . Alternatively, we also use a shorthand $\mathbb{I}_k(a(i))$ to define an indicator function evaluating the proposition $a(i) = k$. Finally, we denote by $\mathcal{C}_i^K = \{a(i) \mid i \in \mathcal{N}_i^K\}$ the set of agents having access to (at least one) of the K -hop neighbors of the data point i . We underline that \mathcal{N}_i^K and \mathcal{C}_i^K are different: the first one is a set of data points, whereas the latter is a set of agents. For instance, all the neighbors of datum i can be on the same agent k , or they can be found across multiple agents (see Fig. 1). This makes the design of a fully-distributed strategy especially challenging due to the coupling between data and communication graphs. Naturally, we have that $\mathcal{C}_i^1 \subseteq \mathcal{C}_i^2 \subseteq \dots \subseteq \mathcal{C}_i^K$.

B. Distributed GCN Inference

To devise a distributed GCN, we need to rewrite the basic layer (1) in a way that is amenable for a fully-distributed implementation. Then, stacking multiple layers of this form will naturally provide us with a distributed GCN model. In particular, the operation of a GC layer is composed of a local component (the right-most multiplication by \mathbf{W}), and a

¹Recall that we use indexes k and z to refer to an agent, while indexes i and j to refer to data nodes, like in the previous section.

²Whenever this is not valid, we might remove the edge from the original data graph, eventually re-normalizing the weighted adjacency matrix. For some data graphs (e.g., similarity graphs) this is feasible, and we evaluate the effect on performance in Sec. V-C.

communication step that is coherent with the communication topology under our assumption. However, the matrix \mathbf{W} is a global parameter, and it is unrealistic to assume that all agents have access to it in the distributed scenario that we are considering in this work. Thus, we relax the model by assuming that each agent has its own set of weights \mathbf{W}_k , i.e., a local copy of the global variable \mathbf{W} (see. Sec. III.C for distributed training of the local weights). In such a case, the i -th output row of the GC layer in (1) reads as:

$$\mathbf{h}_i = \phi \left(\sum_{j \in \mathcal{N}_i} D_{ij} [\mathbf{W}_{a(j)}^T \mathbf{x}_j] \right), \quad (5)$$

where \mathcal{N}_i is the (one-hop) inclusive neighborhood of vertex i . For a GC layer in the form (5), the i -th output depends on the agents in the set \mathcal{C}_i^1 . The question is: What messages should be sent across the agents' network for computing layer (5)? To evaluate (5) at node i , the agent $k = a(i)$ must receive the following messages from agents $z \in \mathcal{C}_i^1$:

$$\sum_{z \in \mathcal{N}_i} \mathbb{I}_z(a(j)) \cdot D_{iz} [\mathbf{W}_z^T \mathbf{x}_j]. \quad (6)$$

In particular, denoting by $B_{kz} = \sum_{(i,j) \in \mathcal{E}_D} \mathbb{I}_k(a(i)) \mathbb{I}_z(a(j))$ the number of data points that share a connection from agent z to agent k (e.g., $B_{12} = 2$ in Fig. 1), and assuming that the GC layer inputs and outputs have size H , the two agents z and k must exchange $B_{kz}H$ values for the inference phase.

Stacking multiple layers in the form (5), we obtain a fully-distributed version of a GCN. For example, the i -th output of a two-layer GCN can be written as:

$$\hat{\mathbf{y}}_i = \phi \left(\sum_{l \in \mathcal{N}_i} D_{il} \mathbf{V}_{a(l)}^T \phi \left(\sum_{j \in \mathcal{N}_i} D_{lj} [\mathbf{W}_{a(j)}^T \mathbf{x}_j] \right) \right). \quad (7)$$

The expression in (7) depends on \mathbf{V}_k for $k \in \mathcal{C}_i^1$, but also on \mathbf{W}_k for $k \in \mathcal{C}_i^2$. Thus, increasing the number of layers of the GCN, we enlarge the set of multi-hop neighbors of node i in the data graph, thus potentially increasing the number of agents participating in the prediction (7). Note that (7) can be implemented by two sequential communication steps over the network of agents, both exploiting messages in the form (6).

C. Distributed GCN training

To train the GCN in a distributed fashion, we need to solve (3) exploiting only message-passing over the communication graph. Let $\mathbf{w}_k \in \mathbb{R}^p$ be the vector comprising all trainable parameters of agent k (e.g., $\mathbf{w}_k = [\text{vec}(\mathbf{W}_k); \text{vec}(\mathbf{V}_k)]$ in (7)). Then, we reformulate (3) introducing a consensus constraint that forces all the local copies $\{\mathbf{w}_k\}_{k=1}^m$ to be equal:

$$\begin{aligned} \arg \min_{\mathbf{w}_1, \dots, \mathbf{w}_m} \quad & \frac{1}{|\mathcal{T}_D|} \sum_{i \in \mathcal{T}_D} l(\mathbf{w}_1, \dots, \mathbf{w}_m; \hat{\mathbf{y}}_i, y_i) \\ \text{s.t.} \quad & \mathbf{w}_1 = \mathbf{w}_2 = \dots = \mathbf{w}_m. \end{aligned} \quad (8)$$

Inspired by the methods in [29]–[31], we proceed minimizing (8) by means of a distributed projected gradient algorithm, which we will explain in the sequel.

Letting $\bar{\mathbf{w}} = [\mathbf{w}_1^T, \dots, \mathbf{w}_m^T]^T$ be the stack of all agents' vectors, the agreement constraint set in (8) can be equivalently rewritten as:

$$\mathcal{S} = \left\{ \bar{\mathbf{w}} \mid \left(\frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T \otimes \mathbf{I}_p \right) \bar{\mathbf{w}} = \bar{\mathbf{w}} \right\},$$

where \otimes is the Kronecker product, $\mathbf{1}_m$ is the m -dimensional vector of all ones, and \mathbf{I}_p is the p -dimensional identity matrix. In other words, the matrix

$$\Pi_{\mathcal{S}} = \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T \otimes \mathbf{I}_p \quad (9)$$

is the orthogonal projection operator onto the consensus space. Then, letting

$$L(\bar{\mathbf{w}}) = \frac{1}{|\mathcal{T}_D|} \sum_{i \in \mathcal{T}_D} l(\bar{\mathbf{w}}; \hat{y}_i, y_i),$$

a centralized projected gradient algorithm would proceed by iteratively minimizing (8) as:

$$\bar{\mathbf{w}}^{t+1} = \Pi_{\mathcal{S}}(\bar{\mathbf{w}}^t - \eta^t \nabla_{\bar{\mathbf{w}}} L(\bar{\mathbf{w}}^t)), \quad (10)$$

where η^t is a (possibly time-varying) step-size sequence.

In principle, the procedure in (10) can be implemented only by a central processor that has access to all the data. Indeed, in first place, the function $L(\bar{\mathbf{w}})$ is not separable among the agents's variables (a key difference with respect to [29]–[31]). However, thanks to the specific coupling induced by the data graph, $\nabla_{\bar{\mathbf{w}}} L(\bar{\mathbf{w}}^t)$ in (10) can still be computed in a distributed fashion. To see this, let $\nabla_{\mathbf{w}_k} L(\bar{\mathbf{w}}^t)$ be the gradient component of $\nabla_{\bar{\mathbf{w}}} L(\bar{\mathbf{w}}^t)$ associated with agent k . This k -th sub-component can be computed locally by exchanging data only within the neighborhood of agent k , with the aim of performing a distributed backpropagation step (cf. Fig. 1). In particular, considering a GCN with K layers, the set of all agents involved in the computation of $\nabla_{\mathbf{w}_k} L(\bar{\mathbf{w}}^t)$ is given by:

$$\bar{\mathcal{C}}_k = \bigcup_{i \in \mathcal{T}_k} \mathcal{C}_i^K, \quad (11)$$

which means that

$$\nabla_{\mathbf{w}_k} L(\bar{\mathbf{w}}^t) = \nabla_{\mathbf{w}_k} L(\{\mathbf{w}_z^t\}_{z \in \bar{\mathcal{C}}_k}). \quad (12)$$

The equality in (12) can be seen readily from the definition of $\bar{\mathcal{C}}_k$ in (11). Consider for example an order-1 GC layer in the form (5), and an agent $z \notin \bar{\mathcal{C}}_k$. Then, by the definition in (4), we have that $D_{ij} = 0$ in (5) and (6) whenever $a(i) = k$ and $a(j) = z$, so that no component in L can depend on \mathbf{w}_z^t . Similarly to the inference phase (see Sec. III.B), the computation of (12) requires (possibly sequential) communication steps, where network agents send messages that are symmetric to those in (6), as can be seen by explicitly differentiating (7) [or (5)]. Note that, in this phase, nodes without labeled examples will act in a passive way, simply diffusing the required messages across the network.

Now, even if the gradient step can be computed in a distributed fashion, the projection step performed in (10) through the multiplication by the matrix $\Pi_{\mathcal{S}}$ is inherently a global averaging step from all agents. In the distributed setting that we consider in this work, there is no central agent that

can perform such data fusion step. Thus, to find a distributed implementation capable of solving problem (8), we substitute the projection operator $\Pi_{\mathcal{S}}$ in (10) with a sparse combination matrix \mathbf{C} , whose sparsity pattern reflects the topology of the communication graph. In the sequel, we illustrate the properties of the combination matrix \mathbf{C} .

Assumption A: The matrix $\mathbf{C} \in \mathbb{R}^{m \times m}$ satisfies the following properties:

- (A1) The matrix is symmetric and doubly stochastic, i.e., $\mathbf{1}^T \mathbf{C} = \mathbf{1}^T$ and $\mathbf{C} \mathbf{1} = \mathbf{1}$;
- (A2) $\rho \left(\mathbf{C} - \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T \right) \leq 1 - \gamma < 1$, where $\rho(\mathbf{X})$ denotes the spectral radius of matrix \mathbf{X} .
- (A3) $C_{kz} = 0$ if the edge $(k, z) \notin \mathcal{E}_C$, for all (k, z) , i.e., the matrix reflects the connectivity between agents.

Assumption (A2) ensures that the communication graph is connected, and information can propagate all over the network. Indeed, w.l.o.g., any combination matrix can be written as $\mathbf{C} = \mathbf{I} - \mathbf{L}$, where \mathbf{L} is a weighted Laplacian matrix. By construction, \mathbf{C} has a constant eigenvector associated with the unitary eigenvalue, whose multiplicity is equal to the number of connected components of the graph [29]. Thus, A2 states that, once we remove the constant eigenvector (i.e., the consensus subspace) from \mathbf{C} , the spectral radius of the resulting matrix is strictly less than one, thus ensuring graph connectivity and stability of the consensus recursion [18]. Then, under (A1) and (A2), the matrix \mathbf{C} satisfies:

$$\lim_{t \rightarrow \infty} \mathbf{C}^t = \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T, \quad (13)$$

i.e., the iterative application of the combination matrix \mathbf{C} performs in the limit the orthogonal projection onto the agreement set \mathcal{S} . In conclusion, the proposed distributed training algorithm has the following structure:

$$\bar{\mathbf{w}}^{t+1} = (\mathbf{C} \otimes \mathbf{I}_p) \left(\bar{\mathbf{w}}^t - \eta^t \nabla_{\bar{\mathbf{w}}} L(\bar{\mathbf{w}}^t) \right). \quad (14)$$

Recursion (14) can now be implemented in a fully-distributed fashion. Then, exploiting (12), and letting

$$\psi_k^t = \mathbf{w}_k^t - \eta^t \nabla_{\mathbf{w}_k} L(\{\mathbf{w}_z^t\}_{z \in \bar{\mathcal{C}}_k}) \quad (15)$$

be the result of the k -th component of the gradient step in (14), the update of the local estimate at agent k is given by:

$$\mathbf{w}_k^{t+1} = \sum_{z \in \mathcal{N}_k} C_{kz} \psi_z^t, \quad (16)$$

where \mathcal{N}_k is the inclusive neighborhood of agent k in the communication graph (including agent k itself).

The main steps of the proposed distributed training strategy for GCNs are summarized in Algorithm 1.

A note on privacy: In a distributed optimization context, it is generally required that the algorithm is privacy-preserving, i.e., no input data is ever transparently exposed over the network. While the consensus step in (16) trivially ensures this property, this is slightly more critical for the inference of the first layer, in which the agents exchange linear combinations of the data points. While the matrices \mathbf{W}_k are not necessarily invertible, this could still pose a non-negligible privacy threat

Algorithm 1 : Distributed GCN training and inference

Data: Number of iterations T ; matrix \mathbf{C} satisfying (A1)-(A3); step-size sequence η^t .

- 1: Randomly initialize $\mathbf{w}_1^0, \dots, \mathbf{w}_m^0$.
- 2: **for** $t \in \{0, \dots, T-1\}$, all agents in parallel **do**
- 3: Evaluate the prediction \hat{y}_i for all labeled data.
- 4: Compute the local estimate ψ_k^t using (15).
- 5: $\mathbf{w}_k^{t+1} = \sum_{z \in \mathcal{N}_k} C_{kz} \psi_z^t$
- 6: **end for**

whenever the two agents share a large number of connections. We leave an investigation of differentially-private [57] versions of this algorithm for a future work.

D. Convergence Analysis

In this section, we illustrate the convergence of the proposed distributed training algorithm to stationary solutions of (3).

Proposition 1: A point $\bar{\mathbf{w}}^* \in \mathcal{S}$ is a stationary solution of Problem (8) if a gradient $\nabla L(\bar{\mathbf{w}}^*)$ exists such that³

$$\Pi_S \nabla L(\bar{\mathbf{w}}^*) = \mathbf{0}. \quad (17)$$

Proof. From the minimum principle, any stationary solution $\bar{\mathbf{w}}^*$ of (8) must satisfy

$$\bar{\mathbf{w}}^* = \Pi_S(\bar{\mathbf{w}}^* - \nabla L(\bar{\mathbf{w}}^*)). \quad (18)$$

Since $\Pi_S \bar{\mathbf{w}}^* = \bar{\mathbf{w}}^*$, (18) leads immediately to (17). ■

We consider the following assumptions on function $L(\bar{\mathbf{w}}) = \sum_{i \in \mathcal{T}_D} l(\bar{\mathbf{w}}; \hat{y}_i, y_i)$ in problem (8).

Assumption B [On the objective function]: The objective function L in (8) is continuous and satisfies:

(B1) L is a differentiable, nonconvex function, with Lipschitz continuous gradient, i.e.,

$$\|\nabla L(\mathbf{w}) - \nabla L(\mathbf{z})\| \leq c_L \|\mathbf{w} - \mathbf{z}\|, \text{ for all } \mathbf{w}, \mathbf{z};$$

(B2) L has bounded gradients, i.e., there exists $G > 0$ such that $\|\nabla L(\mathbf{w})\| \leq G$ for all \mathbf{w} ;

(B3) L is coercive, i.e., $\lim_{\|\mathbf{w}\| \rightarrow \infty} L(\mathbf{w}) = +\infty$.

Assumption B is standard and satisfied by many practical problems, see, e.g., [18]–[20], [25]. In particular, assumption (B3), together with the continuity of L , guarantees the existence of a global minimizer of problem (8). Also, we consider two alternative choices for the step-size sequence $\{\eta^t\}_t$ in Algorithm 1, which are illustrated in the following assumption.

Assumption C [On the step-size]: The step-size sequence $\{\eta^t\}_t$ is chosen as:

(C1) a constant, i.e., $\eta^t = \eta > 0$ for all t ;

(C2) a diminishing sequence such that $\eta^t > 0$, for all t ,

$$\sum_{t=0}^{\infty} \eta^t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} (\eta^t)^2 < \infty.$$

Now, letting $\bar{\mathbf{w}}_S^t = \Pi_S \bar{\mathbf{w}}^t$ be the orthogonal projection of the sequence $\{\bar{\mathbf{w}}^t\}_t$ onto \mathcal{S} , we define the performance metric

$$g^t = \|\nabla L(\bar{\mathbf{w}}_S^t)\|_{\Pi_S}^2 = \nabla L(\bar{\mathbf{w}}_S^t)^T \Pi_S \nabla L(\bar{\mathbf{w}}_S^t), \quad (19)$$

³To ease the notation, from now on we will use ∇L instead of $\nabla_{\bar{\mathbf{w}}} L$.

which quantifies proximity to a stationary solution of (8) [cf. (17)]. Also, let

$$g_{best}^t = \inf_{n=1, \dots, t} g^n = \inf_{n=1, \dots, t} \|\nabla L(\bar{\mathbf{w}}_S^n)\|_{\Pi_S}^2. \quad (20)$$

We now illustrate the convergence properties of the proposed distributed training strategy, which are summarized in the following Theorem.

Theorem 2: Let $\{\bar{\mathbf{w}}^t\}_t$ be the sequence generated by Algorithm 1, and let $\bar{\mathbf{w}}_S^t = \Pi_S \bar{\mathbf{w}}^t$ be its orthogonal projection onto \mathcal{S} . Suppose that conditions (A1)-(A3) and (B1)-(B3) hold. Then, the following results hold.

(a) [*Consensus*]: Under (C1), we have:

$$\lim_{t \rightarrow \infty} \|\bar{\mathbf{w}}^t - \bar{\mathbf{w}}_S^t\| = O(\eta); \quad (21)$$

if (C2) holds, the sequence $\{\bar{\mathbf{w}}^t\}_t$ asymptotically converges to the consensus subspace \mathcal{S} , i.e.,

$$\lim_{t \rightarrow \infty} \|\bar{\mathbf{w}}^t - \bar{\mathbf{w}}_S^t\| = 0; \quad (22)$$

(b) [*Convergence*]: Under (C1), we obtain:

$$\lim_{t \rightarrow \infty} g_{best}^t = \mathcal{O}(\eta), \quad (23)$$

with convergence rate $\mathcal{O}\left(\frac{1}{t+1}\right)$; finally, if (B2) holds, we have:

$$\lim_{t \rightarrow \infty} g^t = 0, \quad (24)$$

i.e., $\{\bar{\mathbf{w}}_S^t\}_t$ converges to a stationary solution of (8). ■

Proof. See Appendix B. ■

E. Extension to order- P GCN

The GCNs we considered up to now can be interpreted as a sequence of order-1 filters, interleaved with pointwise nonlinearities. More generic order- P GC layers generalize this setup, and they can be shown to outperform their order-1 counterparts [56], [58], [59]. Extending (1), an order- P GC layer can be written as [56]:

$$\mathbf{H} = \phi \left(\sum_{p=0}^P \mathbf{D}^p \mathbf{X} \mathbf{W}_p \right), \quad (25)$$

where P is a hyper-parameter, and $\mathbf{W}_0, \dots, \mathbf{W}_P$ are separate trainable parameters. While the i th output in (25) depends on neighbors up to P hops away, the layer is still implementable as P iterative rounds of message passing across the graph. As a result, it is immediate to extend our setup to include them. Consider for example a distributed version of an order-2 GC layer, where agent z is endowed with three local sets of parameters $\mathbf{W}_{p,z}$, $p = 0, 1, 2$. Define for simplicity $\tilde{\mathbf{x}}_{p,i} = \mathbf{W}_{p,a(i)}^T \mathbf{x}_i$. Then, the distributed layer can be written as:

$$\mathbf{h}_i = \phi \left(\tilde{\mathbf{x}}_{0,i} + \sum_{j \in \mathcal{N}_i} D_{ij} \left[\tilde{\mathbf{x}}_{1,j} + \sum_{l \in \mathcal{N}_j} D_{jl} \tilde{\mathbf{x}}_{2,l} \right] \right), \quad (26)$$

where $\tilde{\mathbf{x}}_{1,j}$ and $\tilde{\mathbf{x}}_{2,l}$ are the messages to be exchanged across the graph. Similarly, we can concatenate multiple layers of the form (26) to obtain distributed order-2 GCNs. In this case, the i th output of L order- P layers will depend on the set \mathcal{C}_i^{PL} .

IV. MATCHING DATA AND COMMUNICATION GRAPHS

To implement Algorithm 1, the communication network between the agents (described by \mathbf{C}) must satisfy three constraints: (i) it must be connected (even when the underlying data graph is disconnected) [cf. (A2)]; (b) the matrix \mathbf{C} must be symmetric and doubly stochastic [cf. (A1)]; and (c) the graph must allow communication between two agents whenever they share at least one edge in the data graph, in order to enable the proposed distributed inference and training.

Recall from Section III that we denote with B_{kz} the number of data edges in common between agents k and z . Then, let us denote by $\mathbf{A} \in \{0, 1\}^{m \times m}$ a Boolean matrix such that:

$$A_{kz} = \begin{cases} 1 & \text{if } B_{kz} = 0; \\ 0 & \text{otherwise, and if } k = z. \end{cases} \quad (27)$$

Thus, the set of all feasible matrices \mathbf{C} is described by:

$$\mathcal{X} = \left\{ \mathbf{C} \mid \mathbf{C}\mathbf{1} = \mathbf{1}, \mathbf{C}^T = \mathbf{C}, \rho \left(\mathbf{C} - \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T \right) < 1, C_{kz} \neq 0 \text{ if } A_{kz} = 0 \right\}. \quad (28)$$

We can identify two extremes in the set \mathcal{C} . On one hand, we can connect only agents that share at least one connection between data points (the sparsest possible connectivity). On the other hand, we can consider fully-connected networks, where every pair of agents is connected. Of course, connectivity has also an impact on the convergence speed of the proposed distributed procedure. Thus, in general, we would like a principled way to optimize the communication topology given the constraints on the data graph and a target convergence speed of the consensus mechanism in (16). To this aim, we propose the following optimization problem to design the matrix \mathbf{C} (and, consequently, the communication topology of the agents' network):

$$\begin{aligned} \arg \min_{\mathbf{C}} \quad & \|\mathbf{C} \odot \mathbf{A}\|_1 \\ \text{s.t.} \quad & \mathbf{C} = \mathbf{C}^T, \quad \mathbf{C}\mathbf{1} = \mathbf{1}, \\ & \rho \left(\mathbf{C} - \frac{1}{m} \mathbf{1}\mathbf{1}^T \right) \leq 1 - \gamma \end{aligned} \quad (29)$$

where \odot denotes the Hadamard product, $\|\cdot\|_1$ is the ℓ_1 norm on the elements of the matrix, and γ is a hyper-parameter to be chosen. The rationale underlying (29) is to impose sparsity on the links that are not necessary to train the distributed GCN model (i.e., those such that $A_{kz} = 1$ for all $k \neq z$), while enforcing conditions (A1) and (A2) on matrix \mathbf{C} . In particular, the last constraint in (29) ensures the network connectivity, and controls the convergence speed of the consensus step in (16). In particular, lower values of $1 - \gamma$ lead to larger graph connectivity and, consequently, to faster diffusion of information over the network. Thus, solving (29) while varying γ enables us to explore multiple feasible connectivities between the two extremes discussed above, while guaranteeing convergence of the proposed algorithm for distributed GCN training. Note that this problem must be solved only once, before deploying the distributed GCN model.

Problem (29) is convex, and can be solved using standard numerical tools based on semi-definite programming (SDP) [60]. However, it is known that such methods might become extremely inefficient if the number of variables (in this case, the number m of agents) becomes too large, e.g., of the order of hundreds of agents. Thus, in the sequel, we propose an efficient iterative method based on the alternating direction method of multipliers (ADMM) [15] to solve (29).

ADMM solution

Let us denote the feasible set of (29) as:

$$\tilde{\mathcal{X}} = \left\{ \mathbf{C} \mid \mathbf{C}\mathbf{1} = \mathbf{1}, \mathbf{C}^T = \mathbf{C}, \rho \left(\mathbf{C} - \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T \right) \leq 1 - \gamma \right\}.$$

Then, let us consider the equivalent problem formulation of (29), which writes as:

$$\begin{aligned} \arg \min_{\mathbf{C}, \mathbf{Z}} \quad & \|\mathbf{Z} \odot \mathbf{A}\|_1 + \mathbb{I}_{\tilde{\mathcal{X}}}(\mathbf{C}), \\ \text{s.t.} \quad & \mathbf{C} = \mathbf{Z} \end{aligned} \quad (30)$$

where $\mathbb{I}_{\tilde{\mathcal{X}}}(\cdot)$ is the indicator function with respect to the feasible constraint set $\tilde{\mathcal{X}}$. Now, we exploit the scaled version of the ADMM algorithm [15], which alternates among the following three optimization steps. First, letting \mathbf{U}^t be the estimate at time t of the (scaled) Lagrange multiplier associated with the equality constraint in (30), we solve a regularized problem with respect to \mathbf{C} , given by:

$$\mathbf{C}^{t+1} = \arg \min_{\mathbf{C}} \left\{ \mathbb{I}_{\tilde{\mathcal{X}}}(\mathbf{C}) + \frac{1}{2} \|\mathbf{C} - \mathbf{Z}^t + \mathbf{U}^t\|_F^2 \right\}. \quad (31)$$

The step in (31) can be computed in closed-form as:

$$\mathbf{C}^{t+1} = \mathcal{P}_{\tilde{\mathcal{X}}}(\mathbf{Z}^t - \mathbf{U}^t), \quad (32)$$

where $\mathcal{P}_{\tilde{\mathcal{X}}}(\cdot)$ denotes the orthogonal projection onto the feasible set $\tilde{\mathcal{X}}$. Following similar arguments as in [30], letting $\mathbf{\Pi}_1 = \mathbf{1}_m \mathbf{1}_m^T / m$, the projection of a generic matrix \mathbf{X} onto the set $\tilde{\mathcal{X}}$ is given by:

$$\mathcal{P}_{\tilde{\mathcal{X}}}(\mathbf{X}) = \mathbf{\Pi}_1 + \sum_{i=1}^m \beta_i \mathbf{v}_i \mathbf{v}_i^T, \quad (33)$$

where

$$\beta_i = \begin{cases} -1 + \gamma, & \text{if } \lambda_i < -1 + \gamma; \\ 1 - \gamma, & \text{if } \lambda_i > 1 - \gamma; \\ \lambda_i, & \text{if } |\lambda_i| \leq 1 - \gamma. \end{cases} \quad (34)$$

and $\{\lambda_i, \mathbf{v}_i\}$ denote the eigenvalues and eigenvectors of matrix

$$(\mathbf{I} - \mathbf{\Pi}_1) \left(\frac{\mathbf{X} + \mathbf{X}^T}{2} \right) (\mathbf{I} - \mathbf{\Pi}_1). \quad (35)$$

The second step of scaled ADMM performs the following optimization with respect to the primal variable \mathbf{Z} :

$$\mathbf{Z}^{t+1} = \arg \min_{\mathbf{Z}} \left\{ \|\mathbf{Z} \odot \mathbf{A}\|_1 + \frac{\varrho}{2} \|\mathbf{C}^{t+1} - \mathbf{Z} + \mathbf{U}^t\|_F^2 \right\}, \quad (36)$$

where $\varrho > 0$ is the penalty parameter of the augmented Lagrangian in ADMM. By setting the subgradient of the

Algorithm 2 : ADMM-based solution for (29)

Data: matrix \mathbf{A} ; \mathbf{C}^0 , \mathbf{U}^0 , and \mathbf{Z}^0 chosen at random; $\varrho > 0$. Then, for each time $t \geq 0$, repeat the following steps:

$$(S.1) \quad \mathbf{C}^{t+1} = \mathcal{P}_{\tilde{\mathcal{X}}}(\mathbf{Z}^t - \mathbf{U}^t)$$

where the projector $\mathcal{P}_{\tilde{\mathcal{X}}}(\cdot)$ is given by (33)-(35);

$$(S.2) \quad \mathbf{Z}^{t+1} = \mathcal{S}_{\frac{1}{\varrho}}((\mathbf{C}^{t+1} + \mathbf{U}^t) \odot \mathbf{A})$$

$$+ (\mathbf{C}^{t+1} + \mathbf{U}^t) \odot (\mathbf{1}_{m \times m} - \mathbf{A})$$

$$(S.3) \quad \mathbf{U}^{t+1} = \mathbf{U}^t + \mathbf{C}^{t+1} - \mathbf{Z}^{t+1}$$

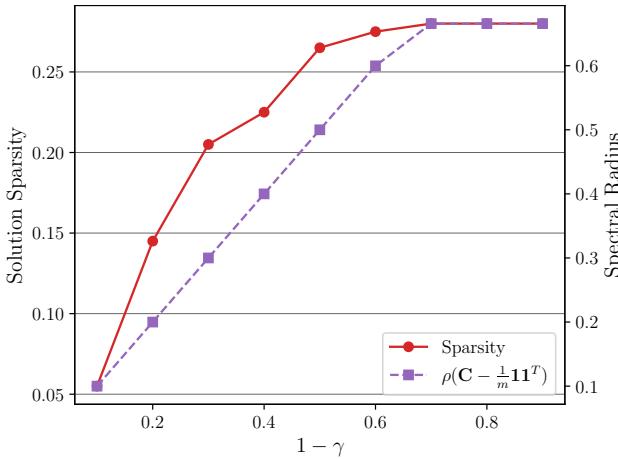


Fig. 2. Sparsity of the final solution (percentage of zero entries) when solving the optimization problem for a varying γ .

objective in (36) to zero, it is straightforward to see that the solution is given by:

$$\begin{aligned} \mathbf{Z}^{t+1} &= \mathcal{S}_{1/\varrho}((\mathbf{C}^{t+1} + \mathbf{U}^t) \odot \mathbf{A}) \\ &+ (\mathbf{C}^{t+1} + \mathbf{U}^t) \odot (\mathbf{1}_{m \times m} - \mathbf{A}), \end{aligned} \quad (37)$$

where $\mathcal{S}_\epsilon(x) = \text{sign}(x) \cdot \max(|x| - \epsilon, 0)$ is the (element-wise) soft-thresholding operator with threshold parameter ϵ . Finally, we perform the multipliers update of scaled ADMM:

$$\mathbf{U}^{t+1} = \mathbf{U}^t + \mathbf{C}^{t+1} - \mathbf{Z}^{t+1}. \quad (38)$$

The main steps of the proposed ADMM solution are summarized in Algorithm 2. The computational complexity of Algorithm 2 is determined by the projection step in (S.1), which requires $O(m^3)$ operations.

As a numerical example, we show in Fig. 2 the sparsity of the matrix \mathbf{C} (defined as the percentage of zero entries) with respect to $1 - \gamma$, when solving the optimization problem (29) for $m = 20$ agents and randomly initialized matrix \mathbf{A} . From Fig. 2, we can notice how the sparsity of the communication graph improves by increasing the value of the hyper-parameter $1 - \gamma$, until it reaches the minimally-connected network corresponding to the perfect matching with the data graph (i.e., $C_{kz} > 0$ i.f.f. $A_{kz} = 0$).

TABLE I
SUMMARY OF THE THREE BENCHMARKS USED IN SECTION V.

Dataset	$ \mathcal{V}_D $	$ \mathcal{E}_D $	$ \mathcal{T}_D $	Classes	Features
CITESEER	2110	3668	120	6	3703
CORA	2810	7981	140	7	2879
PUBMED	19717	44324	60	3	500

V. EXPERIMENTAL EVALUATION

In this section, we extensively assess the performance of the proposed strategy for distributed training (and inference) of graph convolutional networks.

A. Experimental setup

Datasets: We apply our proposed approach to three node classification benchmarks taken from the literature on graph NNs, see [9]. The characteristics of each dataset are briefly summarized in Table I, where we report the size of the underlying data graph, the number of classes for each problem, and the amount of nodes that are in the training subset. The nodes in the three datasets refer to scientific publications. Each node is endowed with a binary feature vector denoting the presence/absence of a set of words in the text. An edge in the graph represents a citation among two publications, and texts must be grouped in one of several categories. A deeper description of the three datasets and their train/test splits can be found in [61].

Architectures: Our baseline is a standard GCN inspired by [9]. It is built of two GC layers as in (2). The hidden layer has 64 units and ReLU activations. A dropout layer with probability 0.5 is inserted after the input and after the hidden layer to stabilize training. As a second comparison, we consider the same GCN, but we set $\mathbf{D} = \mathbf{I}$, thus eliminating any contribution from the graph and obtaining a classical NN. All weight matrices are initialized from a Gaussian distribution with mean zero and standard deviation 10^{-3} . The standard GCN and NN share the same initialization, while for the distributed GCN (dGCN) we initialize a separate weight matrix for each agent.

Agents' Network: We experiment with $5 - 10 - 15 - 20$ agents. Because the three benchmarks have low degree in the data graph [9], randomly partitioning data across the agents results with very high probability in fully-connected networks even for networks having 20 agents. For this reason, we build a more realistic scenario by selecting m seed nodes at random in the data graph, and then expanding their graph randomly in a breadth-first search. In this way, we obtain relatively well clustered assignments, such that sparsity between agents is preserved. To build the agents' connectivity graph \mathbf{C} , we then solve the optimization problem from Section IV with $\gamma = 0.5$.

Experiments: The centralized architectures (i.e., NN and GCN) are trained using gradient descent; whereas, Algorithm 1 is used to train our distributed GCN. In all cases, we used a constant step-size rule, where the learning rate is fine-tuned manually for each dataset/algorithm to obtain the fastest convergence speed. All experiments are averaged 10

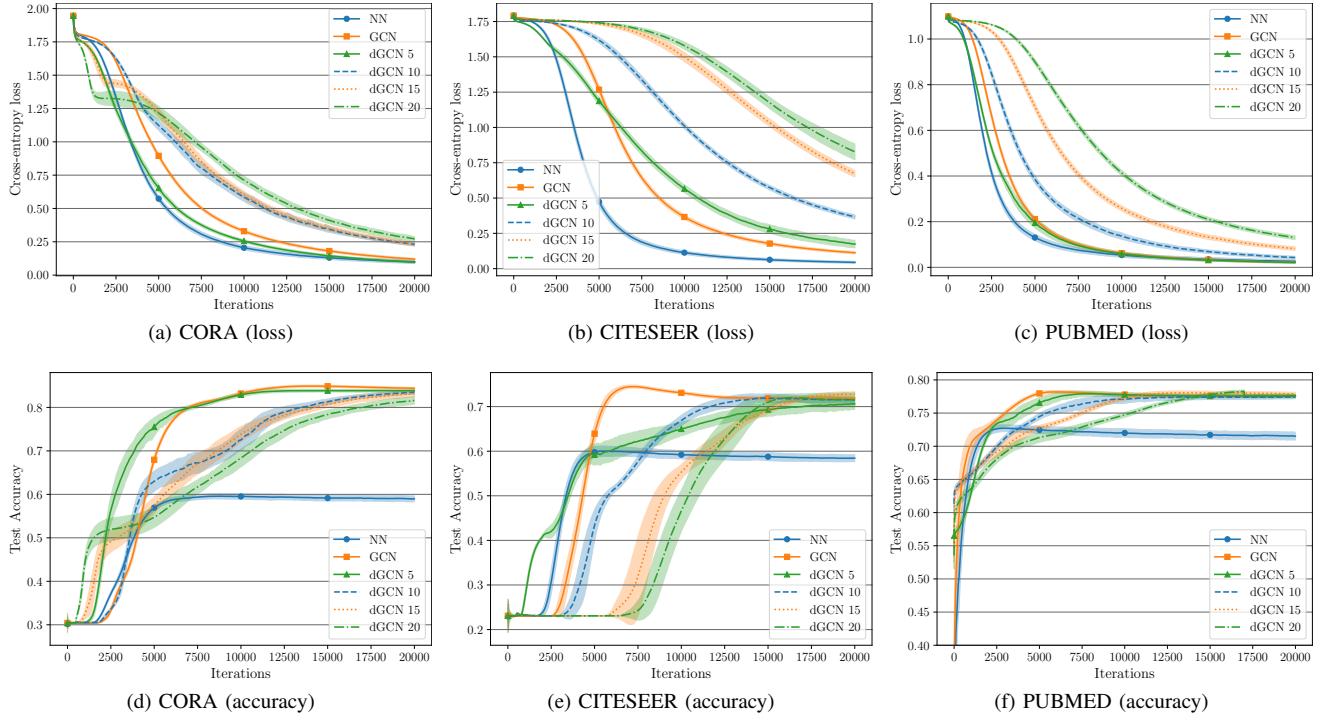


Fig. 3. Results for the proposed approach, compared to a standard GCN and a standard NN. First row are the training losses, second row are the test accuracies. (a)-(d) CORA; (b)-(e) CITESEER; (c)-(f) PUBMED.

times with respect to the initial weights, and figures illustrate both the mean and the standard deviation. All experiments are implemented in JAX [62].

B. Comparisons and discussion

In Fig. 3 we summarize the results of the simulations. Each column represents a dataset, while we show the training losses on the upper row (in log scale) and the test accuracies on the bottom row. The first trivial observation we can make is that the GCN clearly outperforms the standard NN in all three scenarios in terms of test accuracy. While this is far from being novel [9], it is worth underlining it here as the clearest motivation for proposing a distributed protocol for graph convolutional networks. Secondly, we can see that the dGCN asymptotically matches the performance of the centralized GCN when looking at accuracy in all three cases. In term of losses, it matches very closely the performance of GCN for CORA and PUBMED, while we see a slightly slower convergence rate for CITESEER. In Section V-D we look at how to close this gap by modifying the update rule of the proposed approach. In all our experiments, consensus between the parameters belonging to different agents is reached relatively quickly, with the average absolute distance between any two set of parameters decreasing below 10^{-2} in approximately 200-300 iterations. The averaging step of our algorithm (line 5 in Algorithm 1) is essential to guarantee stability, as we show with an ablation study later on in Section V-E.

C. Results with sparse connectivity

Up to now, we assumed that the connectivity between agents matches the connectivity between data points, i.e., (5) is

feasible according to the agents' topology. Next, we evaluate the performance of our method whenever this assumption is relaxed and the connectivity of the agents does not exactly match the connectivity of the underlying data graph (e.g., because of physical constraints in the available communications among agents). To this end, we replicate the experiments of the previous section, but we randomly drop up to 75% of the connections between agents, while ensuring that the agents' graph remains connected overall. In addition, we experiment with ring and line topologies, to show the effect of the most minimal connectivity. After this, we renormalize both the data graph and the agents' graph and run our dGCN. First, we can evaluate the impact of removing all data graph connections that are not consistent with the new agents' connectivity in Fig. 4(a), where we show the number of edges remaining in \mathcal{E}_D after the renormalization step. Note that, even in the most minimal connectivity (the line one), only up to 35% of the original data connections are lost. The results in terms of training loss and test accuracies for the CORA dataset with 10 agents are shown in Fig. 4(b)-(c). As we can notice from Fig. 4, a strong reduction of network connectivity impacts mildly the performance of the proposed strategy. Results for the other two datasets are similar and are omitted for brevity.

D. Results with a different update rule

While we framed our method in the context of a standard gradient descent, the only requirement for convergence is that the local update rule for every agent provides a reasonably good descent direction. To this end, we replace the GD step in (15) with an Adam and a momentum optimizers, where the

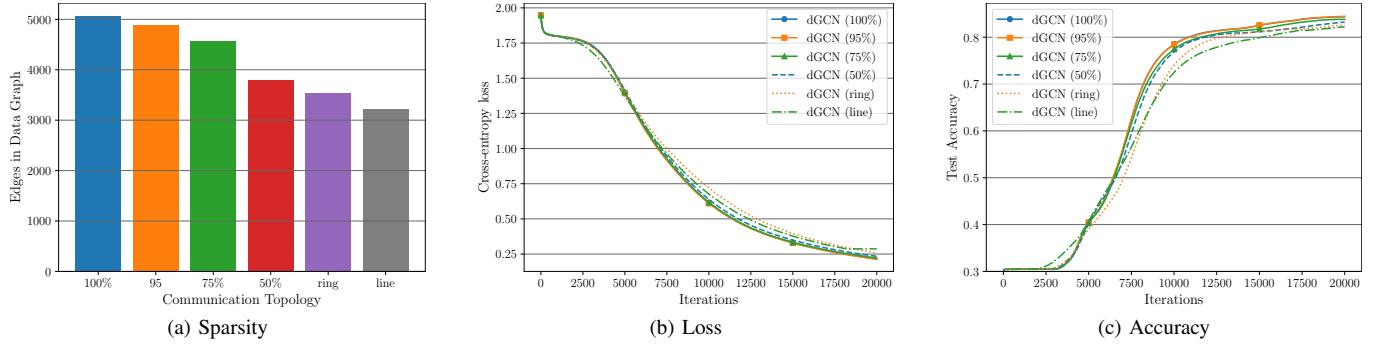


Fig. 4. Results of the proposed approach when the agents' connectivity is sparse. A percentage between brackets denotes how many edges in the agents' graph have been kept, while the remaining ones have been randomly removed. (a) Number of edges remaining in \mathcal{E}_D (for CORA) after removing those not consistent with a sparse agents' connectivity graph. (b) Training loss. (c) Test accuracy.

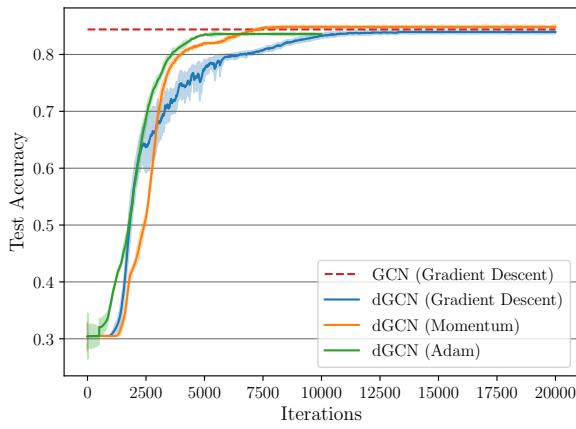


Fig. 5. Results of the proposed approach when replacing the gradient descent step with a momentum optimizer or an Adam-like step (training loss).

additional hyper-parameters are always manually fine-tuned for the quickest convergence.

In our implementation, for Adam every agent keeps track of a local set of first-order and second-order statistics of the data that are used for the update step, while for momentum we simply reuse a running average of the agents' local gradients. We consider a consensus step on the weights after updating the local statistics, although we have found no noticeable difference in interleaving the two. The results are shown in Fig. 5 for a maximum number of 20000 iterations of optimization. We see that both techniques are able to vastly accelerate the algorithm compared to a naive gradient descent. This is especially important in a distributed context, where each iteration requires exchanging data over the network.

E. Ablation study

In this paragraph, we illustrate a simple ablation study on the consensus step (16) (i.e., step 5 in Algorithm 1). In particular, we evaluate the results of the algorithm when multiple gradient descent steps are executed before each consensus step, and with an ablated version where no consensus is ever performed. The results are shown in Fig. 6. From 6, we can see that running a consensus step every iteration is required for reaching the highest accuracy. In fact, long intervals with

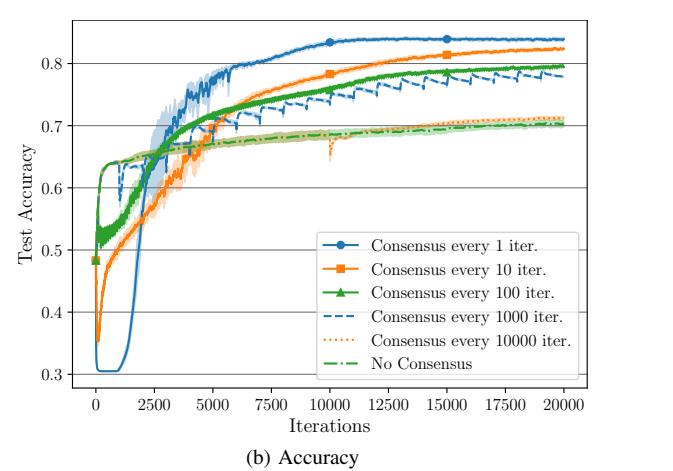
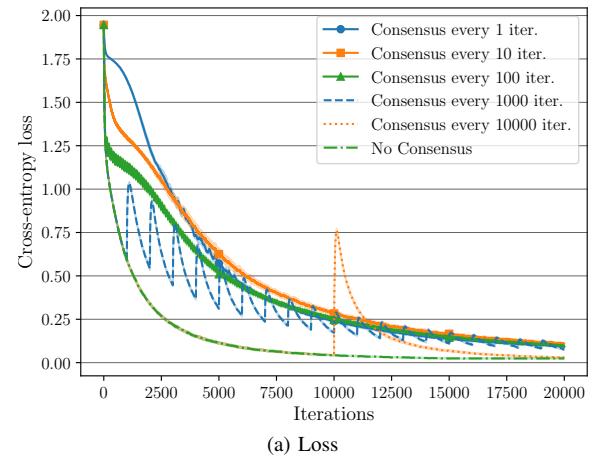


Fig. 6. Results of the proposed approach when considering consensus steps every x gradient descent steps. (a) Training loss. (b) Test accuracy.

no consensus tend to destabilize the training phase, while removing the consensus step quickly makes all agents go into a strong overfitting phase. This last behaviour can be understood as each agent minimizing the loss only on its own local portion of the training data graph, with minimal influence from the outside parts (i.e., the parts belonging to other agents). In fact, we expect the importance of the consensus constraint to grow as the ratio between the number of agents and the overall size of the data graph decreases.

TABLE II
RESULTS OF THREE DISTRIBUTED ALGORITHMS ON THE TRAFFIC PREDICTION BENCHMARKS FROM SECTION V-F

Dataset	Linear	GCN	Order-2 GCN
PeMSD4	0.231 ± 0.001	0.148 ± 0.002	0.126 ± 0.001
PeMSD8	0.266 ± 0.001	0.185 ± 0.002	0.147 ± 0.003

F. Experiments on a traffic prediction scenario

We conclude the experimental section validating our model on two highway traffic prediction datasets (PeMSD4 and PeMSD8) collected by the Caltrans Performance Measurement System (PeMS) of California [53]. The measurement system collects data from more than 39000 sensors every 30 seconds. PeMSD4 aggregates the data in January and February 2018 of the San Francisco Bay Area, while PeMSD8 aggregates the data from July to September 2016 in San Bernardino. We use the same preprocessing procedure from [63]. In particular, we resample the data at 5 minutes interval, remove any sensor that is distant more than 3.5 km from any other sensor, and fill any missing value using linear interpolation. We also perform the same train/test split as in [63]. The adjacency data matrix is given by $A_{ij} = \exp(-d_{ij}^2/10)$, with d_{ij} being the distance between two sensors [64]. The matrix is then sparsified by removing all elements below 0.5 [64].

We assume that each sensor sends its measurements to the closest among 6 base stations that we artificially place on a ring topology across the area (i.e., we have $m = 6$ agents equispaced on a ring connectivity). For each timestep t , the input \mathbf{x}_i for each sensor is a collection of the last 6 recordings (i.e., the last half hour of measurements). Each sensor is then tasked with predicting the successive measurement. The loss function is computed by averaging the squared error for all timesteps in the training dataset and for all sensors. Then, beyond the distributed GCN of the previous section, we also consider a distributed linear model obtained by removing the hidden layer from the GCN definition, and a distributed order-2 GCN (see Section III-E) implemented with Chebyshev polynomials [48]. We average the results over 5 different initialization of the parameters, and we report the resulting mean-squared error on the test portion of the datasets in Table II. Both distributed neural networks models significantly outperform the distributed linear one, showing the importance of having distributed training protocols for graph neural networks. The order-2 GCN further improves the results, at the cost of requiring approximately 3 times the amount of parameters exchanged over the network (as each layer is now parameterized by three different weight matrices).

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose the first algorithmic framework for distributed training of graph convolutional networks, which can be implemented in the absence of any form of central coordination. The proposed method extends previous works on distributed machine learning to the case data have some relational structure, in the form of pairwise connections between points. The resulting model is naturally distributed

along three different lines: (i) during inference, (ii) during backpropagation, and (iii) during consensus phase.

Because this is one of the first works connecting these two fields of research, several extensions and contributions are possible. On one hand, our algorithm requires three communication steps instead of a single consensus, opening the possibility of compression and/or reduction protocols for the number of messages exchanged across the network. In addition, we plan to evaluate the algorithm on different graph NN variants and problems, especially in situations wherein the agents' connectivity might be time-varying. An open challenge concerns situations in which our assumption of a proper matching between the data graph and the agents' graph is not satisfied. In this case, 'lost' edges in the data graph should be recovered exploiting multi-hop exchanges of information among agents, which will be explored in future works. Finally, an important open research question in graph neural networks is scaling up to larger graphs and streaming data. We believe our approach might represent an important stepping stone in this exciting research direction.

APPENDIX A

A LEMMA ON CONVERGENCE OF SCALAR SEQUENCES

We introduce a lemma dealing with convergence of some sequences, which will be used in the following arguments.

Lemma 3: Let $0 < \zeta < 1$, and let $\{\alpha^t\}$ and $\{\nu^t\}$ be two positive scalar sequences. Then, the following hold:

(a) If $\lim_{t \rightarrow \infty} \alpha^t = 0$, then

$$\lim_{t \rightarrow \infty} \sum_{n=1}^t \zeta^{t-n} \alpha^n = 0. \quad (39)$$

(b) If $\sum_{t=1}^{\infty} (\alpha^t)^2 < \infty$ and $\sum_{t=1}^{\infty} (\nu^t)^2 < \infty$, then

$$\lim_{t \rightarrow \infty} \sum_{n=1}^t \sum_{l=1}^n \zeta^{n-l} \alpha^n \nu^l < \infty. \quad (40)$$

APPENDIX B

PROOF OF THEOREM 2

Point (a): Let $\Pi_{\mathcal{S}^\perp} = \mathbf{I} - \Pi_{\mathcal{S}}$ be the projector onto the subspace orthogonal to \mathcal{S} . We will now study the temporal evolution of

$$\bar{\mathbf{w}}_{\mathcal{S}^\perp}^t = \Pi_{\mathcal{S}^\perp} \bar{\mathbf{w}}^t = \bar{\mathbf{w}}^t - \Pi_{\mathcal{S}} \bar{\mathbf{w}}^t = \bar{\mathbf{w}}^t - \bar{\mathbf{w}}_{\mathcal{S}}^t \quad (41)$$

i.e., the component of $\bar{\mathbf{w}}^t$ that lies in the subspace orthogonal to \mathcal{S} . To this aim, multiplying (14) from the left side by $\Pi_{\mathcal{S}^\perp}$, and letting $\mathbf{c}^t = \Pi_{\mathcal{S}^\perp}(\mathbf{C} \otimes \mathbf{I}_p)\nabla L(\bar{\mathbf{w}}^t)$, we obtain:

$$\Pi_{\mathcal{S}^\perp} \bar{\mathbf{w}}^{t+1} = \Pi_{\mathcal{S}^\perp}(\mathbf{C} \otimes \mathbf{I}_p) \bar{\mathbf{w}}^t - \eta^t \mathbf{c}^t. \quad (42)$$

Now, since $\Pi_{\mathcal{S}^\perp}(\mathbf{C} \otimes \mathbf{I}_p) = \Pi_{\mathcal{S}^\perp}(\mathbf{C} \otimes \mathbf{I}_p)\Pi_{\mathcal{S}^\perp}$ if (A1) holds, eq. (42) becomes:

$$\begin{aligned} \bar{\mathbf{w}}_{\mathcal{S}^\perp}^{t+1} &= \Pi_{\mathcal{S}^\perp}(\mathbf{C} \otimes \mathbf{I}_p) \bar{\mathbf{w}}_{\mathcal{S}^\perp}^t - \eta^t \mathbf{c}^t \\ &= \left(\left(\mathbf{C} - \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T \right) \otimes \mathbf{I}_p \right) \bar{\mathbf{w}}_{\mathcal{S}^\perp}^t - \eta^t \mathbf{c}^t. \end{aligned} \quad (43)$$

Iterating recursion (43), we have:

$$\begin{aligned}\bar{\mathbf{w}}_{\mathcal{S}^\perp}^t &= \left(\left(\mathbf{C} - \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T \right) \otimes \mathbf{I}_p \right)^t \bar{\mathbf{w}}_{\mathcal{S}^\perp}^0 \\ &\quad - \sum_{l=1}^t \left(\left(\mathbf{C} - \frac{1}{m} \mathbf{1}_m \mathbf{1}_m^T \right) \otimes \mathbf{I}_p \right)^{t-l} \eta^{l-1} \mathbf{c}^{l-1}.\end{aligned}\quad (44)$$

Now, taking the norm of (44) and using (B2), (A2), we obtain:

$$\|\bar{\mathbf{w}}_{\mathcal{S}^\perp}^t\| \leq (1-\gamma)^t \|\bar{\mathbf{w}}_{\mathcal{S}^\perp}^0\| + G \sum_{l=1}^t (1-\gamma)^{t-l} \eta^{l-1}. \quad (45)$$

Then, if (C1) holds, taking the limit of (45), since $(1-\gamma)^t \rightarrow 0$

as $t \rightarrow \infty$ and $\lim_{t \rightarrow \infty} \sum_{l=1}^t (1-\gamma)^{t-l} = \frac{1}{\gamma}$, we have:

$$\lim_{t \rightarrow \infty} \|\bar{\mathbf{w}}_{\mathcal{S}^\perp}^t\| \leq \frac{G\eta}{\gamma} = O(\eta), \quad (46)$$

which proves (21) [cf. (41)]. On the other side, if (C2) holds, invoking Lemma 3(a) [cf. (39)], from (45) we conclude that:

$$\lim_{t \rightarrow \infty} \|\bar{\mathbf{w}}_{\mathcal{S}^\perp}^t\| = 0, \quad (47)$$

thus proving also (22), and completing the proof of point (a).

Point (b): We now prove the convergent behavior of $\bar{\mathbf{w}}_{\mathcal{S}}^t$, i.e., the component of $\bar{\mathbf{w}}^t$ that lies in \mathcal{S} . Thus, multiplying (14) from the left side by $\Pi_{\mathcal{S}}$, and using (A1), we obtain:

$$\bar{\mathbf{w}}_{\mathcal{S}}^{t+1} = \bar{\mathbf{w}}_{\mathcal{S}}^t - \eta^t \Pi_{\mathcal{S}} \nabla L(\bar{\mathbf{w}}^t). \quad (48)$$

Now, under (A1), invoking the descent lemma on L and using (48), we obtain:

$$\begin{aligned}L(\bar{\mathbf{w}}_{\mathcal{S}}^{t+1}) &\leq L(\bar{\mathbf{w}}_{\mathcal{S}}^t) - \eta^t \nabla L(\bar{\mathbf{w}}_{\mathcal{S}}^t)^T \Pi_{\mathcal{S}} \nabla L(\bar{\mathbf{w}}^t) \\ &\quad + \frac{c_L}{2} (\eta^t)^2 \|\Pi_{\mathcal{S}} \nabla L(\bar{\mathbf{w}}^t)\|^2.\end{aligned}\quad (49)$$

Now, summing and subtracting the vector $\nabla L(\bar{\mathbf{w}}_{\mathcal{S}}^t)$ properly in the second term on the RHS of (49), we obtain:

$$\begin{aligned}L(\bar{\mathbf{w}}_{\mathcal{S}}^{t+1}) &\stackrel{(a)}{\leq} L(\bar{\mathbf{w}}_{\mathcal{S}}^t) - \eta^t \|\nabla L(\bar{\mathbf{w}}_{\mathcal{S}}^t)\|_{\Pi_{\mathcal{S}}}^2 + \frac{c_L}{2} G^2 (\eta^t)^2 \\ &\quad - \eta^t \nabla L(\bar{\mathbf{w}}_{\mathcal{S}}^t)^T \Pi_{\mathcal{S}} (\nabla L(\bar{\mathbf{w}}^t) - \nabla L(\bar{\mathbf{w}}_{\mathcal{S}}^t)) \\ &\stackrel{(b)}{\leq} L(\bar{\mathbf{w}}_{\mathcal{S}}^t) - \eta^t g^t + \frac{c_L}{2} G^2 (\eta^t)^2 + \eta^t c_L G \|\bar{\mathbf{w}}^t - \bar{\mathbf{w}}_{\mathcal{S}}^t\| \\ &\stackrel{(c)}{\leq} L(\bar{\mathbf{w}}_{\mathcal{S}}^t) - \eta^t g^t + \frac{c_L}{2} G^2 (\eta^t)^2 + c_L G \|\bar{\mathbf{w}}_{\mathcal{S}^\perp}^0\| (1-\gamma)^t \eta^t \\ &\quad + c_L G^2 \eta^t \sum_{l=1}^t (1-\gamma)^{t-l} \eta^{l-1},\end{aligned}\quad (50)$$

where in (a) we used (B2); (b) comes from (B1), (B2), and (19); and in (c) we used (45) [cf. (41)]. Now, applying recursively (50), since under (B3) L is bounded from below (w.l.o.g., we consider $L(\bar{\mathbf{w}}_{\mathcal{S}}^{t+1}) \geq 0$ for all t), we obtain:

$$\begin{aligned}\sum_{n=0}^t \eta^n g^n &\leq L(\bar{\mathbf{w}}_{\mathcal{S}}^0) + c_L G \|\bar{\mathbf{w}}_{\mathcal{S}^\perp}^0\| \sum_{n=0}^t (1-\gamma)^n \eta^n \\ &\quad + \frac{c_L}{2} G^2 \sum_{n=0}^t (\eta^n)^2 + c_L G^2 \sum_{n=0}^t \sum_{l=1}^n (1-\gamma)^{n-l} \eta^n \eta^{l-1}.\end{aligned}\quad (51)$$

Using $\sum_{n=0}^t \eta^n g^n \geq \left(\sum_{n=0}^t \eta^n \right) g_{best}^t$ in (51) [cf. (20)], we get:

$$\begin{aligned}g_{best}^t &\leq \left(L(\bar{\mathbf{w}}_{\mathcal{S}}^0) + c_L G \|\bar{\mathbf{w}}_{\mathcal{S}^\perp}^0\| \sum_{n=0}^t (1-\gamma)^n \eta^n + \frac{c_L}{2} G^2 \sum_{n=0}^t (\eta^n)^2 \right. \\ &\quad \left. + c_L G^2 \sum_{n=0}^t \sum_{l=1}^n (1-\gamma)^{n-l} \eta^n \eta^{l-1} \right) \frac{1}{\left(\sum_{n=0}^t \eta^n \right)}.\end{aligned}\quad (52)$$

Then, if (C1) holds, exploiting $\sum_{n=0}^t (1-\gamma)^n \leq 1/\gamma$ for all t , from (52) we obtain:

$$\begin{aligned}g_{best}^t &\leq L(\bar{\mathbf{w}}_{\mathcal{S}}^0) + \frac{c_L G \|\bar{\mathbf{w}}_{\mathcal{S}^\perp}^0\| \eta}{\gamma} + \frac{c_L}{2} (t+1) G^2 \eta^2 \\ &\quad + \frac{c_L G^2 \eta^2}{\gamma} (t+1) \frac{1}{(t+1)\eta}.\end{aligned}\quad (53)$$

Thus, from (53), it is easy to see that

$$\lim_{t \rightarrow \infty} g_{best}^t \leq \frac{c_L G^2 (2+\gamma)}{2\gamma} \eta = O(\eta), \quad (54)$$

with convergence rate $O\left(\frac{1}{t+1}\right)$, thus proving (23).

Finally, using (C2) in (52), and exploiting (39) and (40), we directly obtain:

$$\lim_{t \rightarrow \infty} g_{best}^t = \liminf_{t \rightarrow \infty} \|\nabla L(\bar{\mathbf{w}}_{\mathcal{S}}^t)\|_{\Pi_{\mathcal{S}}} = 0. \quad (55)$$

Using similar arguments as in [65, p.1887] (omitted due to lack of space), we can also prove:

$$\limsup_{t \rightarrow \infty} \|\nabla L(\bar{\mathbf{w}}_{\mathcal{S}}^t)\|_{\Pi_{\mathcal{S}}} = 0. \quad (56)$$

In conclusion, from (55) and (56), we must have:

$$\lim_{t \rightarrow \infty} \|\nabla L(\bar{\mathbf{w}}_{\mathcal{S}}^t)\|_{\Pi_{\mathcal{S}}} = 0, \quad (57)$$

i.e., the sequence $\{\bar{\mathbf{w}}_{\mathcal{S}}^t\}_t$ converges to a stationary point of (8) [cf. (17)]. This concludes the proof of Theorem 2.

REFERENCES

- [1] M. E. Newman, D. J. Watts, and S. H. Strogatz, "Random graph models of social networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. suppl 1, pp. 2566–2572, 2002.
- [2] T. Jombart, R. Eggo, P. Dodd, and F. Balloux, "Reconstructing disease outbreaks from genetic data: a graph approach," *Heredity*, vol. 106, no. 2, pp. 383–390, 2011.
- [3] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.
- [4] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proc. 6th International Conference on Learning Representations (ICLR)*, 2018.
- [5] D. Romero, M. Ma, and G. B. Giannakis, "Kernel-based reconstruction of graph signals," *IEEE Transactions on Signal Processing*, vol. 65, no. 3, pp. 764–778, 2017.
- [6] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an, "Graph convolutional encoders for syntax-aware neural machine translation," in *Proc. 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.
- [7] A. B. Lee, D. Luca, L. Klei, B. Devlin, and K. Roeder, "Discovering genetic ancestry using spectral graph theory," *Genetic Epidemiology*, vol. 34, no. 1, pp. 51–59, 2010.
- [8] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *Proc. International Conference on Learning Representations (ICLR)*, 2014.

- [9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *Proc. International Conference on Learning Representations (ICLR)*, 2017.
- [10] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *IEEE Signal Process. Magazine*, vol. 34, no. 4, pp. 18–42, 2017.
- [11] U. Kang, H. Tong, and J. Sun, "Fast random walk graph kernel," in *Proc. 2012 SIAM International Conference on Data Mining (SDM)*. SIAM, 2012, pp. 828–838.
- [12] R. Kondor and H. Pan, "The multiscale laplacian graph kernel," in *Advances in Neural Information Processing Systems*, 2016, pp. 2990–2998.
- [13] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, no. Nov, pp. 2399–2434, 2006.
- [14] D. Cai, X. He, J. Han, and T. S. Huang, "Graph regularized nonnegative matrix factorization for data representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1548–1560, 2010.
- [15] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.
- [16] H. Ouyang, N. He, L. Tran, and A. Gray, "Stochastic alternating direction method of multipliers," in *Proc. 30th International Conference on Machine Learning (ICML)*, 2013, pp. 80–88.
- [17] F. S. Cattivelli, C. G. Lopes, and A. H. Sayed, "Diffusion recursive least-squares for distributed estimation over adaptive networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 5, pp. 1865–1877, 2008.
- [18] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, p. 48, 2009.
- [19] P. Bianchi and J. Jakubowicz, "Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization," *IEEE Transactions on Automatic Control*, vol. 58, no. 2, pp. 391–405, 2012.
- [20] P. Di Lorenzo and G. Scutari, "Next: In-network nonconvex optimization," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 2, pp. 120–136, 2016.
- [21] S. Vlaski and A. H. Sayed, "Distributed learning in non-convex environments—part i: Agreement at a linear rate," *arXiv preprint arXiv:1907.01848*, 2019.
- [22] ———, "Distributed learning in non-convex environments—part ii: Polynomial escape from saddle-points," *arXiv preprint arXiv:1907.01849*, 2019.
- [23] Y. Yang, G. Scutari, D. P. Palomar, and M. Pesavento, "A parallel decomposition method for nonconvex stochastic multi-agent optimization problems," *IEEE Transactions on Signal Processing*, vol. 64, no. 11, pp. 2949–2964, 2016.
- [24] S. Scardapane and P. Di Lorenzo, "A framework for parallel and distributed training of neural networks," *Neural Networks*, vol. 91, pp. 42–54, 2017.
- [25] J. George, T. Yang, H. Bai, and P. Gurram, "Distributed stochastic gradient method for non-convex problems with applications in supervised learning," *arXiv preprint arXiv:1908.06693*, 2019.
- [26] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.
- [27] A. H. Sayed, "Adaptation, learning, and optimization over networks," *Foundations and Trends® in Machine Learning*, vol. 7, no. 4–5, pp. 311–801, 2014.
- [28] A. Nedic, A. Ozdaglar, and P. A. Parrilo, "Constrained consensus and optimization in multi-agent networks," *IEEE Transactions on Automatic Control*, vol. 55, no. 4, pp. 922–938, 2010.
- [29] P. Di Lorenzo, S. Barbarossa, and S. Sardellitti, "Distributed signal processing and optimization based on in-network subspace projections," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2061–2076, 2020.
- [30] R. Nassif, S. Vlaski, and A. H. Sayed, "Adaptation and learning over networks under subspace constraints—part i: Stability analysis," *IEEE Transactions on Signal Processing*, vol. 68, pp. 1346–1360, 2020.
- [31] ———, "Adaptation and learning over networks under subspace constraints part ii: Performance analysis," *IEEE Transactions on Signal Processing*, 2020.
- [32] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.
- [33] M. Tsitsvero, S. Barbarossa, and P. Di Lorenzo, "Signals on graphs: Uncertainty principle and sampling," *IEEE Transactions on Signal Processing*, vol. 64, no. 18, pp. 4845–4860, 2016.
- [34] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs: Frequency analysis," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3042–3054, 2014.
- [35] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," *IEEE Transactions on Signal Processing*, vol. 65, no. 2, pp. 274–288, 2016.
- [36] J. Liu, E. Isufi, and G. Leus, "Filter design for autoregressive moving average graph filters," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 1, pp. 47–60, 2018.
- [37] P. Di Lorenzo, P. Banelli, E. Isufi, S. Barbarossa, and G. Leus, "Adaptive graph signal processing: Algorithms and optimal sampling strategies," *IEEE Transactions on Signal Processing*, vol. 66, no. 13, pp. 3584–3598, 2018.
- [38] P. Di Lorenzo, P. Banelli, S. Barbarossa, and S. Sardellitti, "Distributed adaptive learning of graph signals," *IEEE Transactions on Signal Processing*, vol. 65, no. 16, pp. 4193–4208, 2017.
- [39] M. Coutino, E. Isufi, and G. Leus, "Advances in distributed graph filtering," *IEEE Transactions on Signal Processing*, vol. 67, no. 9, pp. 2320–2333, 2019.
- [40] R. Nassif, C. Richard, J. Chen, and A. H. Sayed, "Distributed diffusion adaptation over graph signals," in *Proc. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4129–4133.
- [41] F. Huo, R. Nassif, C. Richard, H. Wang, and A. H. Sayed, "Online distributed learning over graphs with multitask graph-filter models," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 6, pp. 63–77, 2020.
- [42] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.
- [43] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.
- [44] D. Bacciu, F. Errica, A. Micheli, and M. Podda, "A gentle introduction to deep learning for graphs," *arXiv preprint arXiv:1912.12693*, 2019.
- [45] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.
- [46] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.
- [47] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.
- [48] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, "From graph filters to graph neural networks," *IEEE Signal Processing Magazine*, 2020.
- [49] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, "Pytorch-biggraph: A large-scale graph embedding system," in *Proc. Conference on Systems and Machine Learning (SysML)*, 2019.
- [50] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: a comprehensive graph neural network platform," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2094–2105, 2019.
- [51] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.
- [52] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.
- [53] C. Chen, K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia, "Freeway performance measurement system: Mining loop detector data," *Transportation Research Record*, vol. 1748, no. 1, pp. 96–102, 2001.
- [54] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. 6th International Conference on Learning Representations (ICLR)*, 2018.
- [55] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. 7th International Conference on Learning Representations (ICLR)*, 2019.
- [56] E. Isufi, F. Gama, and A. Ribeiro, "Edgenets: Edge varying graph neural networks," *arXiv preprint arXiv:2001.07620*, 2020.
- [57] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 308–318.

- [58] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, “Pitfalls of graph neural network evaluation,” *arXiv preprint arXiv:1811.05868*, 2018.
- [59] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, “Benchmarking graph neural networks,” *arXiv preprint arXiv:2003.00982*, 2020.
- [60] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
- [61] Z. Yang, W. W. Cohen, and R. Salakhutdinov, “Revisiting semi-supervised learning with graph embeddings,” in *Proc. 33rd International Conference on Machine Learning (ICML)*, 2016, pp. 40–48.
- [62] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, “JAX: composable transformations of Python+NumPy programs,” 2018. [Online]. Available: <http://github.com/google/jax>
- [63] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, “Attention based spatial-temporal graph convolutional networks for traffic flow forecasting,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 922–929.
- [64] B. Yu, H. Yin, and Z. Zhu, “Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting,” in *Proc. 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 3634–3640.
- [65] A. Daneshmand, F. Facchinei, V. Kungurtsev, and G. Scutari, “Hybrid random/deterministic parallel algorithms for nonconvex big data optimization,” *IEEE Transactions on Signal Processing*, vol. 63, no. 13, pp. 3914–3929, Aug. 2015.