mathematics

MDPI

Article

Graph-Informed Neural Networks for Regressions on Graph-Structured Data

Stefano Berrone [1,2,3], Francesco Della Santa [1,2,3,*], Antonio Mastropietro [1,2,4], Sandra Pieraccini [3,5] and Francesco Vaccarino [1,2]

[1] Dipartimento di Scienze Matematiche (DISMA), Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy; stefano.berrone@polito.it (S.B.); antonio.mastropietro@polito.it (A.M.); francesco.vaccarino@polito.it (F.V.)
[2] SmartData@PoliTO Center , Politecnico di Torino, 10129 Turin, Italy
[3] Member of the INdAM-GNCS Research Group, 00100 Rome, Italy; sandra.pieraccini@polito.it
[4] Addfor Industriale s.r.l., Via Giuseppe Giocosa 36/38, 10125 Turin, Italy
[5] Dipartimento di Ingegneria Meccanica e Aerospaziale (DIMEAS), Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy
* Correspondence: francesco.dellasanta@polito.it

Abstract: In this work, we extend the formulation of the spatial-based graph convolutional networks with a new architecture, called the graph-informed neural network (GINN). This new architecture is specifically designed for regression tasks on graph-structured data that are not suitable for the well-known graph neural networks, such as the regression of functions with the domain and codomain defined on two sets of values for the vertices of a graph. In particular, we formulate a new graph-informed (GI) layer that exploits the adjacent matrix of a given graph to define the unit connections in the neural network architecture, describing a new convolution operation for inputs associated with the vertices of the graph. We study the new GINN models with respect to two maximum-flow test problems of stochastic flow networks. GINNs show very good regression abilities and interesting potentialities. Moreover, we conclude by describing a real-world application of the GINNs to a flux regression problem in underground networks of fractures.

Keywords: graph neural networks; deep learning; regression on graphs

MSC: 05C21; 65D15; 68T07; 90C35

Citation: Berrone, S.; Della Santa, F.; Mastropietro, A.; Pieraccini, S.; Vaccarino, F. Graph-Informed Neural Networks for Regressions on Graph-Structured Data. *Mathematics* **2022**, *10*, 786. https://doi.org/10.3390/math10050786

Academic Editor: Konstantin Kozlov

Received: 1 February 2022
Accepted: 25 February 2022
Published: 1 March 2022

1. Introduction

Graphs are frequently used to describe and study many different phenomena, such as transportation systems, epidemic- or economic-default spread, electrical circuits, and social interactions; the literature typically refers to the use of graph theory to analyze such phenomena with the term "network analysis" [1].

Recently, new key contributions to network analyses have been proposed by the neural network (NN) community; in particular, deep learning (DL) approaches can be extended to graph-structured data via the so-called graph neural networks (GNNs). The origin of GNNs dates back to the late 2000s [2–4] when their processing was still too computationally expensive [5]. Nonetheless, the huge success of the convolutional neural networks (CNNs) inspired a new family of GNNs, re-defining the notation of convolutions for graph-structured data and developing the graph convolutional networks (GCNs). According to the taxonomy defined in [5], two main families of GCNs can be observed: the spectral-based GCNs [6–8], which are based on the spectral graph theory, and the spatial-based GCNs [9–12], which are based on the aggregation of the neighbor nodes' information. In particular, the spatial-based GCNs are, nowadays, preferred in many applications, thanks to their flexibility and efficiency [5].

Typically, GCNs are used to perform the following tasks on graph data [5]: (i) semi-supervised node regression or classification; (ii) edge classification or link prediction;

and (iii) graph classification. Nonetheless, even if GCNs have been proven to be good instruments to learn graph data, some challenges still exist. The two main challenges for GCNs are [5]: (i) to build deep architectures with good performances; and (ii) to be scalable for large graphs. The first issue is the most problematic one; indeed, the success of DL lies in its depth, but the literature suggests that going deeper into a GCN is not usually beneficial [5]. Moreover, experimental results for the spectral-based GCNs showed that performances dropped considerably as the number of graph convolutional layers increased [13].

In this work, we present a new type of spatial-based graph convolutional layer designed for regression tasks on graph-structured data, a framework for which previous GCNs are not well suited. Given a graph G with n nodes, a regression task on graph-structured data based on G consists of approximating a function $\boldsymbol{F} : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}^m$, $m \leq n$, depending on the adjacency matrix of G, and that returns the m values related to a fixed subset of m nodes for each set of values assigned to the nodes of G. This type of regression task has applications in many interesting fields, such as circulation with demand (CwD) problems (see [chap. 7.7] in [14]), network interdiction models (NIMs) [15], and flux regression problems in underground fractured media [16,17]. A classic multi-layer perceptron (MLP), or its suitable variants, can perform this regression task on the graph data with a good performance [16,17], implicitly learning the node relationships during the training (see [18,19]). On the other hand, the current GCNs in the literature are not comparable to MLPs for such a regression task; indeed, as mentioned above, they are designed mainly for other kinds of tasks and, in practice, they cannot exploit deep architectures. Then, the idea is to define a new graph convolutional layer that exploits the graph structure to improve the training of the NN (compared to an MLP), and that makes it possible to build deep NN architectures. The new convolution operation for graph data that we define is closer to the convolution of CNNs (see [chap. 9] in [20]) than the convolution of all the other GCNs. Nonetheless, similarities with the classic NN4G layers in [3] and the diffusion-convolutional neural networks (DCNNs) in [21] exist.

Put simply, the simplest version of our graph layer is characterized by a filter with one weight w_i associated with each graph node v_i. Then, the output feature of a node is computed by summing up the input features of the node itself and of its neighbors, where each one is multiplied by the corresponding node weights. We call this new type of graph layer a graph-informed (GI) layer. Indeed, given a scalar value w_j associated with each graph vertex v_j, a GI layer looks like a fully-connected (FC) layer where, for each unit v_i connected with a unit v_j of the previous layer, the weight w_{ji} is equal to 0 if (v_j, v_i) is not an edge of the graph and $i \neq j$, otherwise $w_{ji} = w_j$ (see Equation (4) in Section 2).

Numerical experiments have shown the potentiality of the GI layers, which involves training deep NNs made up of a sequence of GI layers. We define these NNs as graph-informed neural networks (GINNs). In particular, the numerical experiments showed that GINNs were characterized by improved regression abilities with respect to MLPs, thanks also to their ability to overcome the depth problem typical of the other GCNs.

The work is organized as follows: in Section 2, the GI layers are formally introduced and defined, explaining their fundamental operations, properties, similarities, and differences with respect to other spatial-based graph convolutional layers. Section 3 is dedicated to the numerical experiments; in particular, we analyze the regression abilities of the GINNs on a maximum-flow regression problem and we compare the results with the performances obtained on the same problem with MLPs. We conclude the section with a real-world application, studying the application of GINNs to a flux regression problem in underground networks of fractures. In Section 4, we summarize the results and draw some conclusions.

2. Mathematical Formulation of the Graph-Informed Layers

In this section, we describe the mathematical formulation of the new GI layers, based on the adjacency matrix of a graph. In particular, we describe the mathematical details that define the function $\mathcal{L}^{GI}$, describing the action of a GI layer L^{GI}. From now on, we will call

the function describing the action of a generic NN layer as the characterizing function of the layer.

Definition 1 (Graph-Informed layer: basic form)**.** *Let $A \in \mathbb{R}^{n\times n}$ be the adjacency matrix characterizing a given graph $G = (V, E)$ without self-loops, and let $\widehat{A}$ be the matrix $\widehat{A} := A + \mathbb{I}_n$, where $\mathbb{I}_n \in \mathbb{R}^{n\times n}$ is the identity matrix. Then, a graph-informed (GI) layer L^{GI}, with respect to the graph G, is an NN layer with n units connected to a layer with outputs in $\mathbb{R}^n$ with a characterizing function $\mathcal{L}^{GI} : \mathbb{R}^n \to \mathbb{R}^n$ defined by*

$$\mathcal{L}^{GI}(\boldsymbol{x}) = \boldsymbol{f}\Big(\widehat{W}^\top \boldsymbol{x} + \boldsymbol{b}\Big)\,, \tag{1}$$

where:

- *Given a vector $\boldsymbol{w} \in \mathbb{R}^n$ of weights associated with the vertices V, the defined filter of L^{GI} the matrix $\widehat{W}$ is obtained by multiplying the i-th row of $\widehat{A}$ by the weight w_i, i.e.,*

$$\widehat{W} := \mathrm{diag}(\boldsymbol{w})\hat{A}\,, \tag{2}$$

  where $\mathrm{diag}(\boldsymbol{w})$ is the diagonal matrix with a diagonal that corresponds to vector $\boldsymbol{w}$;
- *Given the layer activation function $f : \mathbb{R} \to \mathbb{R}$, we denote by $\boldsymbol{f}$ the element-wise application of f;*
- *$\boldsymbol{b} \in \mathbb{R}^n$ is the vector of biases.*

Broadly speaking, given a directed graph $G = (V, E)$, with n nodes and an adjacency matrix $A \in \mathbb{R}^{n\times n}$, the main idea behind a GI layer is to generalize the convolutional layer filters to the graph-structured features. Indeed, the objective is to endow the layer with the implicit relationship between the features of the adjacent graph nodes, and also to take advantage of the sparse interaction- and parameter-sharing properties typical of convolutional NNs (see [chap. 9.2] in [20]).

Convolutional layers rely on the identification of images as lattices of pixels. The main idea for the GI layer formulation is to adapt convolutional layer concepts to graphs that are not characterized by a lattice structure. We generalize the filter mechanisms of the convolutional layers to the graph-structured data, introducing the concept of graph-based filters. In practice, for each node of the graph G, we consider a weight w_j which is associated to node $v_j \in V$ and we re-define the convolution operation as

$$x'_i = \sum_{j\in \mathrm{N_{in}}(i)\cup\{i\}} x_j\, w_j + b_i\,, \tag{3}$$

where

- x_j denotes the input feature of node $v_j \in V$, for each $j = 1, \ldots, n$;
- $\mathrm{N_{in}}(i)$ is the set of indices j, such that there exists an incoming edge $(v_j, v_i) \in E$;
- b_i is the bias corresponding to node v_i;
- x'_i is the output feature associated to v_i, computed by the filter (see Figure 1).

For a non-directed graph G, Equation (3) does not change, since a non-directed edge $\{v_j, v_i\}$ is equivalent to two directed edges, (v_j, v_i) and (v_i, v_j). Indeed, Definition 1 holds for both directed and non-directed graphs.

Similar to the convolutional layers, which act on the current pixel and on all its neighbors for computing the output feature, in (3) the layer acts on x_i and on the values associated with the incoming neighbors of v_i for the computation of x'_i (see Figure 1). Nonetheless, despite the inspiration received from convolutional layers, a GI layer L^{GI} described by (3) can be seen also as a constrained FC layer, where the weights are such that

$$w_{ji} = \begin{cases} w_j\,, & \text{if } (v_j, v_i) \in E \\ w_i\,, & \text{if } j = i \\ 0\,, & \text{otherwise} \end{cases}, \tag{4}$$

for each $i, j = 1, \ldots, n$.

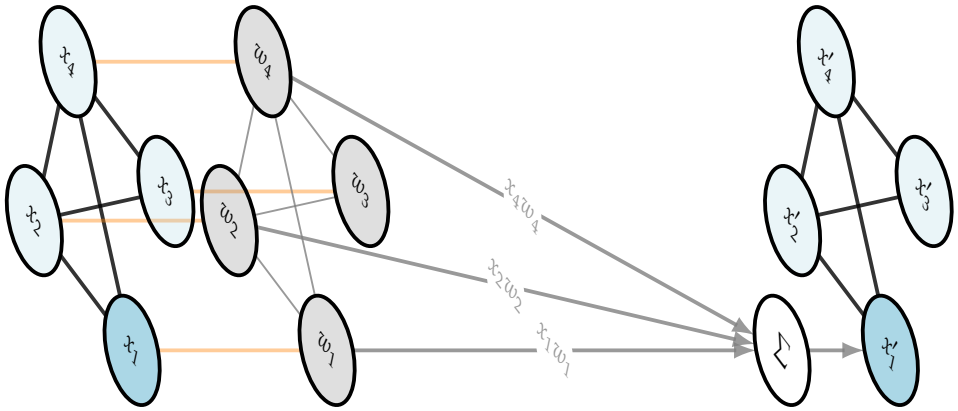

Figure 1. Case of non-directed graph with $n = 4$ nodes. Example of the action of a filter $\boldsymbol{w} \in \mathbb{R}^4$ (grey "layer" of the plot) of a GI layer, applied to feature x_1 of the first graph-node v_1, for the computation of x'_1; for simplicity, the bias is not illustrated. The orange edges describe the multiplication of feature x_i, of node v_i, with the filter's weight w_i, for each $i = 1, \ldots, 4$.

In the next sections, we generalize the action of these kinds of layers to make them able to: (i) receive any arbitrary number $K \geq 1$ of input features for each node; and (ii) return any arbitrary number $F \geq 1$ of output features for each node.

2.1. Generalization to K Input Node Features

Equation (1) describes the simplest case of GI layers, where just one feature is considered for each vertex of the graph for both the inputs and the outputs. We start generalizing the previous definition, taking into account a larger number of features tackled by L^{GI}.

Definition 2 (Graph-Informed layer with K input features per node)**.** *Let $G, A,$ and $\widehat{A}$ be as in Definition 1. Then, a GI layer with $K \in \mathbb{N}$ input features is an NN layer with n units connected to a layer with outputs in $\mathbb{R}^{n\times K}$ with a characterizing function $\mathcal{L}^{GI} : \mathbb{R}^{n\times K} \to \mathbb{R}^n$ defined by*

$$\mathcal{L}^{GI}(X) = \boldsymbol{f}\left(\widetilde{W}^\top \text{vertcat}(X) + \boldsymbol{b}\right), \tag{5}$$

where:

- *$X \in \mathbb{R}^{n\times K}$ is the input matrix (i.e., the output of the previous layer) and* $\text{vertcat}(X)$ *denotes the vector in $\mathbb{R}^{nK}$ obtained by concatenating the columns of X;*
- *Given the matrix $W \in \mathbb{R}^{n\times K}$, the defined filter of L^{GI}, whose columns $\boldsymbol{w}_{\cdot 1}, \ldots, \boldsymbol{w}_{\cdot K} \in \mathbb{R}^n$ are the vectors of weights associated with the k-th input feature of the graph's vertices, the matrix $\widetilde{W} \in \mathbb{R}^{nK\times n}$ is defined as*

$$\widetilde{W} := \begin{bmatrix} \widehat{W}^{(1)} \\ \vdots \\ \widehat{W}^{(K)} \end{bmatrix} = \begin{bmatrix} \text{diag}(\boldsymbol{w}_{\cdot 1})\widehat{A} \\ \vdots \\ \text{diag}(\boldsymbol{w}_{\cdot K})\widehat{A} \end{bmatrix} \in \mathbb{R}^{nK\times n}. \tag{6}$$

The idea behind the generalization from Definitions 1 to 2 is rather simple. Let L be an NN layer with outputs in $\mathbb{R}^{n\times K}$, $K \geq 1$. Therefore, a generic output of L is a matrix $X \in \mathbb{R}^{n\times K}$, whose row $i \in \{1, \ldots, n\}$ describes the K features $x_{i1}, \ldots, x_{iK}$ of node v_i; on the

other hand, each column $\boldsymbol{x}_{\cdot 1}, \ldots, \boldsymbol{x}_{\cdot K}$ of X is equivalent to the output of an NN layer with outputs in $\mathbb{R}^n$.

Therefore, the generalization consists of summing up the action of the K "basic" single-input filters $\boldsymbol{w}_{\cdot 1}, \ldots, \boldsymbol{w}_{\cdot K}$, where each one is applied to $\boldsymbol{x}_{\cdot 1}, \ldots, \boldsymbol{x}_{\cdot K}$, respectively; then, to this sum, we add the bias vector and we apply the activation function. However, this approach is equivalent to (5), i.e., defining one filter W obtained from the concatenation of the basic filters. Indeed:

$$\sum_{k=1}^{K} \widehat{W}^{(k)\top} \boldsymbol{x}_{\cdot k} = \widetilde{W}^{\top} \mathrm{vertcat}(X)\,. \tag{7}$$

Remark 1 (Parallelism with convolutional layers)**.** *It is worth noting that the operations summarized in (5) are an adaptation of the convolutional layer operations to the graph-based inputs. Indeed, the input $X \in \mathbb{R}^{n\times K}$ is equivalent to an $n \times 1$ image with K channels, while $\boldsymbol{w}_{\cdot k}$ is equivalent to the part of the convolutional filter corresponding to the k-th channel of the input image. Then, the output $\mathcal{L}^{GI}(X) \in \mathbb{R}^n$ is equivalent to the so-called activation map of the convolutional layers.*

2.2. Generalization to F Output Node Features

We can further generalize (5) by increasing the number of output features per node returned by the GI layer. This operation is equivalent to building a GI layer characterized by a number $F \geq 1$ of matricial filters, where each one used to compute one of the output features. In a nutshell, the output of these general GI layers is a matrix $Y \in \mathbb{R}^{n\times F}$ whose l-th column $\boldsymbol{y}_{\cdot l} \in \mathbb{R}^n$, $l = 1 \ldots, F$, describes the l-th feature of the nodes of G.

Definition 3 (Graph-Informed layer: general form)**.** *Let $G, A, \widehat{A}$ be as in Definition 1. Then, a GI layer with $K \in \mathbb{N}$ input features and $F \in \mathbb{N}$ output features is an NN layer with nF units connected to a layer with outputs in $\mathbb{R}^{n\times K}$ with a characterizing function $\mathcal{L}_{GI} : \mathbb{R}^{n\times K} \to \mathbb{R}^{n\times F}$ defined by*

$$\mathcal{L}^{GI}(X) = \boldsymbol{f}\left(\widetilde{\mathbf{W}}^{\top} \mathrm{vertcat}(X) + B\right), \tag{8}$$

where:

- *We define the filter of L^{GI}, the tensor $\mathbf{W} \in \mathbb{R}^{n\times K\times F}$, given by the concatenation along the third dimension of the weight matrices $W^{(1)}, \ldots W^{(F)} \in \mathbb{R}^{n\times K}$, corresponding to the F output features of the nodes. Each column $\boldsymbol{w}_{\cdot k}^{(l)} \in \mathbb{R}^n$ of $W^{(l)}$ is the basic filter describing the contribution of the k-th input feature to the computation of the l-th output feature of the nodes, for each $k = 1, \ldots, K$, and $l = 1, \ldots, F$;*
- *The tensor $\widetilde{\mathbf{W}} \in \mathbb{R}^{nK\times F\times n}$ is defined as the concatenation along the second dimension (i.e., the column dimension) of the matrices $\widetilde{W}^{(1)}, \ldots, \widetilde{W}^{(F)}$, such that*

$$\widetilde{W}^{(l)} := \begin{bmatrix} \widehat{W}^{(l,1)} \\ \vdots \\ \widehat{W}^{(l,K)} \end{bmatrix} = \begin{bmatrix} \mathrm{diag}(\boldsymbol{w}_{\cdot 1}^{(l)})\widehat{A} \\ \vdots \\ \mathrm{diag}(\boldsymbol{w}_{\cdot K}^{(l)})\widehat{A} \end{bmatrix} \in \mathbb{R}^{nK\times n}, \tag{9}$$

  for each $l = 1, \ldots, F$. Before the concatenation, the matrices $\widetilde{W}^{(1)}, \ldots, \widetilde{W}^{(F)}$ are reshaped as tensors in $\mathbb{R}^{nk\times 1\times n}$ (see Figure 2);
- *the operation $\widetilde{\mathbf{W}}^{\top}\mathrm{vertcat}(X)$ is a tensor–vector product (see Remark 2);*
- *$B \in \mathbb{R}^{n\times F}$ is the matrix of the biases, i.e., each column $\boldsymbol{b}_{\cdot l}$ is the bias vector corresponding to the l-th output feature of the nodes.*

Notation 1. *From now on, for the sake of simplicity, for each matrix $X \in \mathbb{R}^{n\times K}$, we denote by $\boldsymbol{x}$ the vector* $\mathrm{vertcat}(X) \in \mathbb{R}^{nK}$.

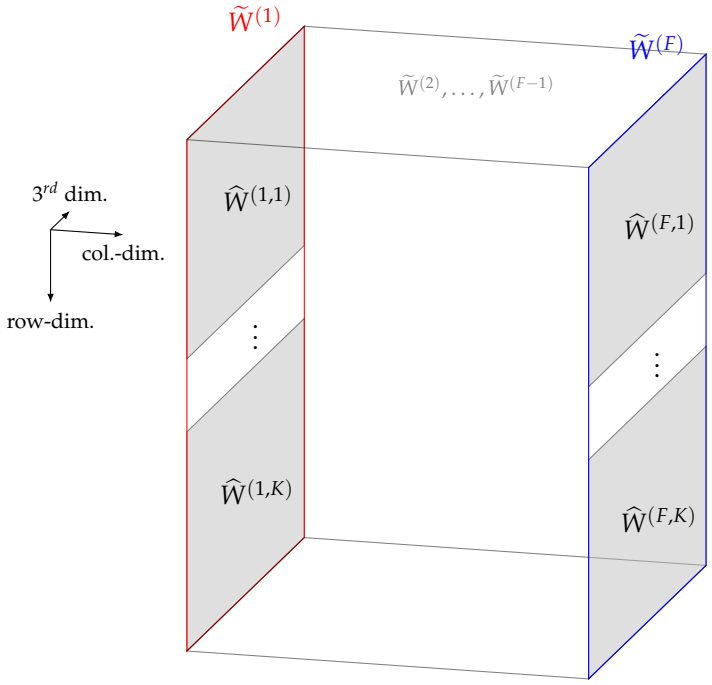

Figure 2. Tensor $\widetilde{\mathbf{W}}$ obtained concatenating along the second dimension of the matrices $\widetilde{W}^{(1)}, \ldots, \widetilde{W}^{(F)} \in \mathbb{R}^{nK \times n}$. Before the concatenation, the matrices are reshaped as tensors in $\mathbb{R}^{nk \times 1 \times n}$.

The generalization of (5) to the case of F output features is built as a function that, for each $X \in \mathbb{R}^{n \times K}$, a matrix is returned $Y \in \mathbb{R}^{n \times F}$ whose l-th column $\boldsymbol{y}_{\cdot l}$, for $l = 1, \ldots, F$, is defined as the application of (5) with respect to a proper filter $W^{(l)} \in \mathbb{R}^{n \times K}$. Indeed, given

$$\boldsymbol{y}_{\cdot l} = \boldsymbol{f}\left(\widetilde{W}^{(l)\top} \boldsymbol{x} + \boldsymbol{b}_{\cdot l}\right) \tag{10}$$

where $\boldsymbol{b}_{\cdot l} \in \mathbb{R}^n$ is the bias vector associated to the l-th filter, we have

$$Y = \begin{bmatrix} \boldsymbol{y}_{\cdot 1} & \cdots & \boldsymbol{y}_{\cdot F} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}\left(\widetilde{W}^{(1)\top} \boldsymbol{x} + \boldsymbol{b}_{\cdot 1}\right) & \cdots & \boldsymbol{f}\left(\widetilde{W}^{(F)\top} \boldsymbol{x} + \boldsymbol{b}_{\cdot F}\right) \end{bmatrix} = \boldsymbol{f}\left(\widetilde{\mathbf{W}}^{\top} \boldsymbol{x} + B\right).$$

Put simply, the generalization to the F output features can be interpreted as a repetition of (5), with respect to F different filters and biases, grouping the results in a matrix Y.

Remark 2. *We recall that the matrix-tensor product of a matrix $M \in \mathbb{R}^{p \times q}$ by a tensor $\mathbf{T} \in \mathbb{R}^{q \times r \times s}$ is given by*

$$M \cdot \mathbf{T} = \mathbf{P} \in \mathbb{R}^{p \times r \times s},$$

where the (i, j, k)-th component p_{ijk} of the tensor $\mathbf{P}$ is defined as

$$p_{ijk} = \sum_{h=1}^{q} m_{ih} t_{hjk},$$

where m_{ih} and t_{hjk} are components of M and $\mathbf{T}$, respectively. Analogously, we can extend this product to tensor–matrix or tensor–tensor pairs.

Moreover, we recall that, for a three-way tensor as $\widetilde{\mathbf{W}}$, the transpose is defined such that the (i, j, k)-th element of $\widetilde{\mathbf{W}}^{\top}$ is equal to the (k, j, i)-th element of $\widetilde{\mathbf{W}}$.

Remark 3 (Total number of parameters)**.** *The total number of parameters in a GI layer, with a characterizing function (8), is $nKF + nF$, i.e., the number of weights plus the number of biases. Let us recall that the number of parameters of a fully-connected layer, with an input shape n and an output shape M is $nM + M$; then, in the case of $M = n$ and $(KF + F) < (n + 1)$, we see that the*

GI layers have a smaller number of parameters to be trained. This observation is important in the case of very large graphs G (i.e., $n \gg 1$).

Remark 4 (GI layer contextualization)**.** *To the best of the authors' knowledge, the GI layers introduced above define a novel typology of spatial GCNs. Equation (1) partially recalls the NN4G layer (see [sec. V.B.] in [3,5]), if one removes both the so-called residual and skip connections (i.e., the extra connections used to directly transfer information between non-consecutive layers). However, the formulation given by the authors in Equation (1) is generalizable to the tensor form (8), i.e., to multiple input/output features, unlike the NN4G layers. It is worth noting that a tensor form , such as (8), is very useful to manage graph-structured regression problems with more than one feature per node.*

Analogously, there are few similarities between the simple L^{GI} layer of Equation (1) and the diffusion-convolutional NNs (DCNNs) of [21] which can be observed, but these GCNs are still different from the GINNs. Indeed, DCNNs are made for different types of tasks, such as node classification or graph classification tasks, inferred from the known features of a subset of nodes. In addition, DCNN layers are based on a degree-normalized transition matrix, computed from the adjacency matrix, "that gives the probability of jumping from node i to node j in one step" [21].

Other similarities between the GINNs and other models can be observed in [22,23], where the adjacency matrix is used to describe the flow of information. Nonetheless, in [22], the NN is built connecting a set of simpler NNs, according to the adjacency matrix. In [23], the interconnected NNs are trained similar to a physics-informed NN (see [24,25]).

In the end, we point the attention of the reader to the fact that, from a theoretical point of view, nothing prevents us from adding a softmax layer at the end of a GINN to extend the new model architecture to cover graph classification tasks with respect to vertex labels (like CNNs for image classification); however, we defer the study of this possibility to future work.

2.3. Additional Properties for GI Layers

The GI layers, in their general formulation (8), can be endowed with additional operations. As is commonly done for the convolutional layers, we add the possibility to endow the GI layers with a pooling operation. However, this operation is different from the one typically used in convolutional layers. Indeed, we define a pooling for GI layers that aggregates the information in the columns of the output matrix, i.e., the values of the F output features of each graph vertex. Given a "reducing" operation (e.g., the mean, the max, the sum, etc.), labeled as rdc, and applied to each row of the matrix returned by (8), the pooling operation for GI layers modifies (8) in the following way:

$$\mathcal{L}^{(GI;\mathrm{rdc})}(X) = \mathrm{rdc}\Big(f\Big(\widetilde{\mathbf{W}}^{\top}\boldsymbol{x} + B\Big)\Big)\,, \tag{11}$$

where rdc is applied row-wise. For example, let $Y \in \mathbb{R}^{n\times F}$ denote the argument of the pooling operation in (11), namely $Y = f\Big(\widetilde{\mathbf{W}}^{\top}\boldsymbol{x} + B\Big)$; the max-pooling operation for a GI layer is such that:

$$\mathcal{L}^{(GI;\max)}(X) = \begin{bmatrix} \max\{y_{11},\ldots,y_{1F}\} \\ \vdots \\ \max\{y_{n1},\ldots,y_{nF}\} \end{bmatrix} \in \mathbb{R}^{n}\,. \tag{12}$$

Note that the pooling operation can be generalized to the application of subgroups to filters, instead of to all the filters. In this case, the pooling operation returns a matrix $Y \in \mathbb{R}^{n\times F'}$, with $F' < F$.

Another operation that is defined for GI layers is the application of a mask on the graph, such that the layer returns values only for a subset $\{v_{i_1},\ldots,v_{i_m}\}$ of the chosen nodes. Let $I = \{i_1,\ldots,i_m\} \subseteq \{1,\ldots,n\}$ label the subset of nodes on which we want to focus the output of the GI layer. Then, a GI layer with a mask operation defined by the set I returns a

sub-matrix $Y' \in \mathbb{R}^{m \times F}$ of the matrix $Y \in \mathbb{R}^{n \times F}$ defined by (8), obtaining extracting rows with index in I; namely,

$$\mathcal{L}^{(GI;I)}(X) = \begin{bmatrix} \left(f\left(\widetilde{\mathbf{W}}^\top \boldsymbol{x} + B\right)\right)_{i_1 \cdot} \\ \vdots \\ \left(f\left(\widetilde{\mathbf{W}}^\top \boldsymbol{x} + B\right)\right)_{i_m \cdot} \end{bmatrix} \in \mathbb{R}^{m \times F}. \tag{13}$$

We end this section with the following proposition, characterizing the relationship between the input and the output features of the graph nodes with respect to a GINN with a subset of H consecutive GI layers. The proof of the statement is straightforward.

Proposition 1 (Number of consecutive GI layers and node interactions)**.** *Let $H \in \mathbb{N}$, $H \geq 1$, be fixed and let $A \in \mathbb{R}^{n \times n}$ be the adjacency matrix of a given graph G. Let us consider a GINN with a subset of H consecutive GI layers $L_1^{GI}, \ldots, L_H^{GI}$, built according to A and with L_h^{GI} connected to L_{h+1}^{GI}, for $h = 1, \ldots, H-1$. Let $d_{ij} := \mathrm{dist}_G(v_i, v_j) \in \mathbb{N} \cup \{+\infty\}$ be the distance between node v_i and node v_j in G. Then, the input feature corresponding to node v_i in L_1^{GI} contributes to the computation of the output feature corresponding to v_j in L_H^{GI} if $H \geq d_{ij}$.*

The proposition above introduces a dependency of a GINN's depth on the complexity of the graph $G = (V, E)$. Let $\boldsymbol{F} : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}^m$ be a function defined on the n vertices of G, and returning a vector of m values associated with the vertices $v_{i_1}, \ldots, v_{i_m} \in V$. Let $\widehat{\boldsymbol{F}} : \mathbb{R}^n \to \mathbb{R}^m$ be the characterizing function of a GINN that approximates $\boldsymbol{F}$. If the output feature of vertex $v_j \in \{v_{i_1}, \ldots, v_{i_m}\}$ through $\boldsymbol{F}$ depends on the input feature of vertex v_i, then the GINN needs at least $d_{ij} = \mathrm{dist}_G(v_i, v_j)$ consecutive GI layers to guarantee that the input feature of v_i contributes in making predictions for the output feature of v_j.

3. Numerical Tests

In this section, we study the potentialities of the GI layers, comparing the regression abilities of GINNs and MLPs for graph-structured data.

The main test problem we consider is the maximum-flow problem: given a flow network, that is, a graph with a source, a sink, and capacities defined on the edges, the goal is to find the maximum flow value that can reach the sink (see [ch. 7.1] in [14]). In particular, we are interested in the stochastic maximum-flow problem, i.e. a problem where the edge capacities are modeled as random variables and the target is to find the distribution of the maximum-flow (e.g., see [26]).

The stochastic maximum-flow problem is a sufficiently general problem for testing GINNs, and it has many interesting applications in network analyses, such as the circulation with demand (CwD) problems (see [ch. 7.7] in [14] and network interdiction models (NIMs) [15]. Put simply, a CwD problem should identify whether or not the maximum flow satisfies a given demand, varying the supply provided by the source and the capacities of the edges; a NIM describes a game in which one or more agents modify the edge capacities to minimize/maximize the maximum flow of the network. These models have many interesting real-world applications, such as the administration of city traffic, the optimization of goods distributions, and identifying vulnerabilities in an operational system.

The section is organized as follows: after a brief description of the maximum-flow problem (Section 3.1), we illustrate the maximum-flow regression problem (Section 3.2). Then, in Section 3.3, we report and compare the performances of the trained GINNs and MLPs. We conclude the section with an example that shows the potentialities of using GINNs in practical applications related to realistic underground flow simulations (Section 3.4).

3.1. The Maximum-Flow Problem

Flow networks are useful models to describe transportation networks, i.e., networks where some sort of traffic flows from a source to a sink along the edges, using the nodes as switches to let the traffic move from an edge to another one (see [ch. 7.1] in [14]). Here, we briefly recall the definition of a flow network.

Definition 4 (Flow Network)**.** *A flow network $\mathcal{G} = (G, s, t, c)$ is a directed graph $G = (V, E)$, of nodes V and edges $E \subseteq V \times V$, such that:*

- *The two nodes $s, t \in V$, $s \neq t$, are defined as the source and the sink of the network, respectively;*
- *c is a real-valued non-negative function defined on the edges, $c : E \to \mathbb{R}_{\geq 0}$, assigning to each edge $e \in E$ a capacity $c_e := c(e)$.*

A flow network $\mathcal{G}$ can be endowed with a flow function.

Definition 5 (Flow)**.** *Let $\mathcal{G}$ be a flow network. An s-t flow (or just flow) on $\mathcal{G}$ is a function*

$$\varphi : E \to \mathbb{R}_{\geq 0}$$

that satisfies the following properties:

- *The capacity condition: for each $e \in E$, it holds $0 \leq \varphi(e) \leq c_e$;*
- *The conservation condition: for each $v \in V \setminus \{s, t\}$, the amount of flow entering v must be equal to the amount of flow leaving v, i.e.,*

$$\sum_{e \in E_{in}(v)} \varphi(e) = \sum_{e \in E_{out}(v)} \varphi(e)\,, \quad \forall\, v \in V \setminus \{s, t\}\,,$$

  where $E_{in}(v) \subset V$ is the subset of the incoming edges of v, and $E_{out}(v) \subset V$ is the subset of outcoming edges of v;
- *The amount of flow leaving the source s must be greater than, or equal to, the one entering s, i.e., $\sum_{e \in E_{in}(s)} \varphi(e) \leq \sum_{e \in E_{out}(s)} \varphi(e)$.*

For the sake of notation, for each $v \in V$ we set

$$\varphi_{\text{in}}(v) = \sum_{e \in E_{\text{in}}(v)} \varphi(e), \quad \varphi_{\text{out}}(v) = \sum_{e \in E_{\text{out}}(v)} \varphi(e), \quad \Delta\varphi_v = \varphi_{\text{out}}(v) - \varphi_{\text{in}}(v)\,,$$

and we call the flow value of a vertex v the quantity $\Delta\varphi_v$.

Then, due to the conservation condition, it holds that $\Delta\varphi_v = 0$, for each $v \in V \setminus \{s, t\}$, and $\Delta\varphi_s \geq 0$. Note that the flow value of the source s is equal to the opposite of the flow value of the sink t, i.e., $\Delta\varphi_t = -\Delta\varphi_s$; for this reason, we refer to $\Delta\varphi_s$ as the flow value of the network.

One of the most common issues concerning a flow network $\mathcal{G}$ is to find a flow that maximizes the effective total flow of the sink t, i.e., to find φ^*, such that

$$\varphi^* = \arg\max_{\varphi} |\Delta\varphi_t|\,.$$

Such a kind of problem is called maximum-flow problem, and it can be solved through the linear programming or many other algorithms (e.g., [27–31]). From a practical point of view, the relationship between the maximum-flow problem and the minimum-cut problem on a flow network $\mathcal{G}$ is particularly important (see [ch. 7.2] in [14] for more details).

Remark 5 (Flow networks and undirected graphs)**.** *Definitions 4 and 5 can be extended to the more complicated case of undirected graphs. Indeed, as observed in Section 2, the non-directed edges $\{v_j, v_i\}$ of a graph are equivalent to the two directed edges (v_j, v_i) and (v_i, v_j). Then, a flow network based on an undirected graph $G = (V, E)$ can be defined as a flow network $\mathcal{G} = (G', s, t, c)$, where*

$G' = (V, E')$ is a directed graph, such that $(u, v), (v, u) \in E'$ if $\{u, v\} \in E$, whose capacity is defined on the edges of G, i.e., $c : E \to \mathbb{R}_{\geq 0}$. As a result, a flow φ defined on such a flow network is a function $\varphi : E' \to \mathbb{R}_{\geq 0}$ characterized by a slightly different capacity condition; namely, φ is such that

$$0 \leq \varphi((u, v)) + \varphi((v, u)) \leq c(\{u, v\}), \quad \forall \{u, v\} \in E.$$

Another approach is to introduce an arbitrary ordering, denoted by "$<$", on the graph nodes and define a directed graph $G' = (V, E')$, such that $(u, v) \in E'$ if $\{u, v\} \in E$ and $u < v$. In this case, a flow φ on the flow network $\mathcal{G} = (G', s, t, c)$, with c defined on the edges E' is a function $\varphi : E' \to \mathbb{R}$, such that the capacity condition is

$$0 \leq |\varphi(e)| \leq c(e), \quad \forall e \in E',$$

and where the entering/exiting behavior of the flows is described by the sign of $\varphi(e)$ and not by the edge direction; i.e., for $(u, v) \in E'$, if $\varphi((u, v)) > 0$ the flow $\varphi((u, v))$ enters in v, whereas if $\varphi((u, v)) < 0$ then $\varphi((u, v))$ enters in u. This latter approach is mainly adopted by software implementations.

3.1.1. The Stochastic Maximum-Flow Problem

The idea of the flow network, flow, and the maximum-flow problem can be easily extended to a stochastic framework, in which edge capacities are modeled as random variables.

Definition 6 (Stochastic flow network)**.** *A stochastic flow network $\mathcal{G} = (G, s, t, p)$ is a directed graph $G = (V, E)$ of nodes V and edges $E \subseteq V \times V$, such that:*

- *The two nodes $s, t \in V$, $s \neq t$, are defined as the source and the sink of the network, respectively;*
- *p is a real-valued non-negative probability distribution for the edge capacities of the network.*

We let $\mathcal{G}(\boldsymbol{c})$ denote the flow network (G, s, t, c) with edge capacities returned by c sampled from p. More specifically, let $e_1, \ldots, e_{|E|}$ be all the edges of G; then $\mathcal{G}(\boldsymbol{c}) = (G, s, t, c)$ if:

- $\boldsymbol{c} \in \mathbb{R}^{|E|}$ is a vector whose c_i is sampled from p;
- The function c is such that $c(e_i) = c_i$, for each $i = 1, \ldots, |E|$.

We denote by $\varphi^{(\boldsymbol{c})}$ a flow defined on the flow network $\mathcal{G}(\boldsymbol{c})$.

The stochastic maximum-flow problem consists of finding the flow

$$\varphi^{*(\boldsymbol{c})} = \arg\max_{\varphi^{(\boldsymbol{c})}} |\Delta\varphi_t^{(\boldsymbol{c})}|, \tag{14}$$

for each fixed vector $\boldsymbol{c}$.

Alternatively, in stochastic maximum-flow problems, one may seek the flow distribution and/or its moments, or the maximum flow value entering the sink t, i.e.,

$$|\Delta\varphi_t^{*(\boldsymbol{c})}| = \max_{\varphi^{(\boldsymbol{c})}} |\Delta\varphi_t^{(\boldsymbol{c})}|, \tag{15}$$

3.2. The Maximum-Flow Regression Problem

A maximum-flow regression problem, with respect to a given stochastic flow network, consists of finding a function that, for each capacity vector $\boldsymbol{c}$, returns an approximation of the maximum flow $|\Delta\varphi_t^{*(\boldsymbol{c})}|$ or an approximation of all the flows reaching the sink t.

Let $\mathcal{G}$ be a stochastic flow network of $n = |E|$ edges and, without a loss of generality, let $e_1, \ldots, e_m \in E$, $m \leq n$ be all the incoming edges of the sink t. Let $\boldsymbol{F} : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}^m$ be a function, such that

$$\boldsymbol{F}(\boldsymbol{c}) = [\varphi^{*(\boldsymbol{c})}(e_1), \ldots, \varphi^{*(\boldsymbol{c})}(e_m)]^\top := \boldsymbol{\varphi}, \tag{16}$$

for each capacity vector $\boldsymbol{c} \in \Omega \subseteq \mathbb{R}^n$ with the elements sampled from the distribution p of the given network $\mathcal{G}$.

From now on, for the sake of simplicity, we drop the dependency from $\boldsymbol{c}$ and the star symbol from the elements of $\boldsymbol{\varphi}$, denoted by $\varphi_1, \ldots, \varphi_m$ the m elements of the vector $\boldsymbol{\varphi} = \boldsymbol{F}(\boldsymbol{c})$. Moreover, assuming the convention of the non-negative flow functions on the graph (see Section 3.1), we denote by φ the ℓ_1-norm of $\boldsymbol{\varphi}$; specifically:

$$\sum_{j=1}^{m} \varphi_i = \sum_{j=1}^{m} |\varphi_i| = \| \boldsymbol{\varphi} \|_1 =: \varphi \,. \tag{17}$$

Then, the target maximum flow with respect to $\boldsymbol{c}$ coincides with φ; indeed, $|\Delta {\varphi_t}^{*(\boldsymbol{c})}| = \sum_{j=1}^{m} \varphi^{*(\boldsymbol{c})}(e_j) = \sum_{j=1}^{m} \varphi_i = \varphi$.

Given the target function $\boldsymbol{F}$ defined by (16), we consider the maximum-flow regression problem with respect to $\mathcal{G}$ looking for an NN with a characterizing function $\widehat{\boldsymbol{F}} : \mathbb{R}^n \to \mathbb{R}^m$, such that $\widehat{\boldsymbol{F}}(\boldsymbol{c})$ approximates $\boldsymbol{F}(\boldsymbol{c})$ for each capacity vector $\boldsymbol{c}$. Namely, setting $\widehat{\boldsymbol{\varphi}} = \widehat{\boldsymbol{F}}(\boldsymbol{c})$ and $\boldsymbol{\varphi} = \boldsymbol{F}(\boldsymbol{c})$, we seek $\widehat{\boldsymbol{\varphi}} \approx \boldsymbol{\varphi}$. To train an NN with respect to $\boldsymbol{F}$, we build a dataset (i.e., a multi-set) $\mathcal{D}$ of pairs $(\boldsymbol{c}\,, \boldsymbol{\varphi}) \in \mathbb{R}^n \times \mathbb{R}^m$, with $\boldsymbol{\varphi} = \boldsymbol{F}(\boldsymbol{c})$, where the capacity vectors are sampled with respect to the distribution p of $\mathcal{G}$; then, $\mathcal{D}$ is split into a training set $\mathcal{T}$, a validation set $\mathcal{V}$, and a test set $\mathcal{P}$ of arbitrary cardinalities. In particular, denoting Θ as the multi-set $\mathcal{T} + \mathcal{V}$, we denote ϑ as the total number of pairs involved in the training operations of the NN, i.e., the pairs in the multi-set Θ ($\vartheta := |\Theta| = |\mathcal{T}| + |\mathcal{V}|$). For more details about multi-sets, see Appendix A.

Once an NN is trained, we evaluate its regression performances by computing two performance measures on the test set $\mathcal{P}$: the edge-wise average mean relative error (MRE_{av}), and the mean relative error on the predicted maxflow (MRE_{φ}). These two errors represent the mean relative error (weighted with respect to the true maxflow) of the predicted flows of the m edges $e_1, \ldots, e_m$ and the mean relative error of the predicted maxflow $\widehat{\varphi} := \sum_{i=1}^{m} \widehat{\varphi}_j$ (i.e., the sum of the elements of $\widehat{\boldsymbol{\varphi}} = \widehat{\boldsymbol{F}}(\boldsymbol{c})$), respectively. For each prediction $\widehat{\boldsymbol{\varphi}}$, let us denote

$$\mathbf{err}(\widehat{\boldsymbol{\varphi}}, \boldsymbol{\varphi}) = [\text{err}_1(\widehat{\boldsymbol{\varphi}}, \boldsymbol{\varphi}), \ldots, \text{err}_m(\widehat{\boldsymbol{\varphi}}, \boldsymbol{\varphi})]^\top := \left[\frac{|\widehat{\varphi}_1 - \varphi_1|}{\varphi}, \ldots, \frac{|\widehat{\varphi}_m - \varphi_m|}{\varphi} \right]^\top$$

as the vector of relative errors computed with respect to the true maxflow $\varphi = \sum_{j=1}^{m} \varphi_j$ (see (17)). Then, the performance measures MRE_{av} and MRE_{φ} are defined as

$$\text{MRE}_{av}(\mathcal{P}) := \frac{1}{m} \sum_{j=1}^{m} \left(\frac{1}{|\mathcal{P}|} \sum_{(\boldsymbol{c}, \boldsymbol{\varphi}) \in \mathcal{P}} \text{err}_j(\widehat{\boldsymbol{\varphi}}, \boldsymbol{\varphi}) \right) \tag{18}$$

and

$$\text{MRE}_{\varphi}(\mathcal{P}) := \frac{1}{|\mathcal{P}|} \sum_{(\boldsymbol{c}, \boldsymbol{\varphi}) \in \mathcal{P}} \frac{|\widehat{\varphi} - \varphi|}{\varphi} \,, \tag{19}$$

respectively.

The smaller both the MRE_{av} and the MRE_{φ} values are on the test set, the better the performances of the NN, with respect to the maximum-flow regression task, are.

Remark 6 (Interpretation of MRE_{av} and MRE_{φ})**.** *It is worth highlighting the different meanings of the errors (18) and (19): MRE_{av} describes the average quality of the NN in predicting the single elements $\varphi_1, \ldots, \varphi_m$ of the target vector $\boldsymbol{\varphi}$, while MRE_{φ} describes the ability of the NN to predict a vector $\widehat{\boldsymbol{\varphi}}$, such that the corresponding maxflow $\widehat{\varphi} = \sum_{j=1}^{m} \widehat{\varphi}_j$ approximates the true maxflow $\varphi = \sum_{j=1}^{m} \varphi_j$. Therefore, a small MRE_{av} corresponds to a good approximation of the flow vectors (i.e., $\widehat{\boldsymbol{\varphi}} \approx \boldsymbol{\varphi}$) and a small MRE_{φ} corresponds to a good approximation of the maximum-flows (i.e., $\widehat{\varphi} \approx \varphi$). Nonetheless, it is important to point out that a small MRE_{av} does not necessarily imply a small MRE_{φ}, and vice-versa. For example, an NN with large MRE_{av}, characterized by the underestimation of the flows φ_{j_1} and by the overestimation of the flows φ_{j_2}, may return a small MRE_{φ} because the sum of the flows is not so far from the true maximum-flow; similarly, a*

large MRE_φ can be obtained from a sufficiently small MRE_{av} if, e.g., the NN underestimates or overestimates all the flows $\varphi_1, \ldots, \varphi_m$ equivalently, such that $\widehat{\boldsymbol{\varphi}} \approx \boldsymbol{\varphi}$ but $\widehat{\varphi} \not\approx \varphi$.

3.2.1. Line Graphs for the Exploitation of GINN Models

Since the inputs of the target function $\boldsymbol{F}$ are the capacity vectors $\boldsymbol{c}$, which are defined on the edges of the graph G and not on the nodes, we need to compute the line graph L of G in order to exploit the GINN models for the maximum-flow regression problem. We recall, here, the definition of line graph (see [32,33]).

Definition 7 (Line Graph). *Let $G = (V, E)$ be a graph (either directed or not). The line graph of G is a graph $L = (E, E')$, such that:*

- *The vertices of L are the edges of G;*
- *Two vertices in L are adjacent if the corresponding edges in G share at least one vertex.*

Given the line graph L of the graph G of a stochastic flow network $\mathcal{G}$, we can use the adjacency matrix A_L of L to define NN models characterized by GI layers to perform the maximum-flow regression task. See the next section for more details about the GINN architectures that are built.

3.3. Maximum-Flow Numerical Experiments

For the experiments related to the maximum-flow regression problem, we take into account two stochastic flow networks:

- $\mathcal{G}_1 = (G_{\text{BA}}, s, t, p)$. The graph G_{BA} of $\mathcal{G}_1$ characterizes a flow network built on an extended Barabási–Albert (BA) model graph [34,35]. Put simply, an extended BA model graph is a random graph generated using a preferential attachment criterion. This family of graphs describes a very common behavior in many natural and human systems, where few nodes are characterized by a higher degree if they are compared to the other nodes of the network.
  In particular, we generate an extended BA undirected graph using the NetworkX Python module [36] (function extended_barabasi_albert_graph, input arguments $\mathtt{n} = 50, \mathtt{m} = 2, \mathtt{p} = 0.15$, and $\mathtt{q} = 0.35$); then, we denote t (the sink of the network) as the node with the highest betweenness centrality [37] and we add a new node s (the source of the network) connected to the 10 nodes with smallest closeness centrality [38,39]. With these operations, we obtain a graph G_{BA} of 51 nodes and $n = |E| = 126$ edges, where the source s is connected to the 10 nodes and the sink t is connected to the $m = 15$ nodes (see Figure 3-left).
  In the end, since, in real-world applications, truncated normal distributions seem to be very common (see Remark 7), in order to simulate a rather general maximum-flow regression problem, we chose a truncated normal distribution between 0 and 10, with a mean of 0, and a standard deviation of $5/3$, as a probability distribution p for the edge capacities (see Section 3.1.1); i.e.,

$$c_i \sim p \equiv \mathcal{N}_{[0,10]}(5\,,\, 5/3)\,, \quad \forall\, i = 1, \ldots, n\,. \tag{20}$$

- $\mathcal{G}_2 = (G_{\text{ER}}, s, t, p)$. The graph G_{ER} of $\mathcal{G}_2$ characterizes a flow network built on an Erdős-Rényi (ER) model graph [40,41]. Put simply, an ER model graph is a random graph generated with a fixed number of nodes, where the edge $e_{ij} = (v_i, v_j)$ has a fixed probability of being created. This family of graphs is typically used to prove and/or find new properties that hold for almost all the graphs; for this reason, we consider a stochastic flow network based on an ER graph in our experiments.
  In particular, we generate an ER undirected graph using the NetworkX Python module [36] (function `fast_gnp_random_graph`, input arguments $\mathtt{n} = 200, \mathtt{p} = 0.01$) and we select its largest connected component G_0 (in terms of the number of vertices). Then, we add to G_0 two new nodes: a node s (the source of the network) connected to

all the nodes with degree equal to 1, and a node t (the sink of the network) connected to the 15 most distant nodes from s. With these operations, we obtain a graph G_{ER} of 171 nodes and $n = |E| = 269$ edges, where the source s is connected to 37 nodes and the sink t is connected to $m = 15$ nodes (see Figure 3-right).
In the end, we chose the truncated normal distribution (20) as the probability distribution p for the edge capacities.

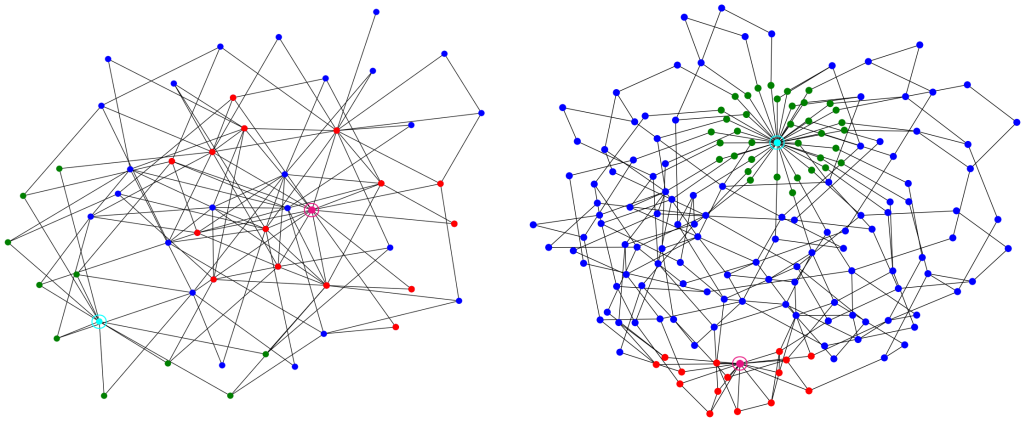

Figure 3. Graph G_{BA} of $\mathcal{G}_1$ (**left**) and graph G_{ER} of $\mathcal{G}_2$ (**right**). In cyan, and with a circle around the source s, in magenta and with a circle around the sink t, in green the nodes connected to s, and in red the nodes connected to t. All the other nodes are in blue.

Remark 7 (Regarding the truncated normal distribution for capacities)**.** *In a network describing a system of highroads, the capacity of a road is defined as $c = k\ell/S$ [42], where $k \in \mathbb{R}^+$ is a value depending on the type of the road, ℓ is the road length, and S is the average distance between two vehicles, typically chosen as a constant value. Then, assuming a network with all roads of the same type (i.e., k constant) and a truncated normal distribution for the length ℓ of the roads, the capacity can be modeled as a random variable with a truncated normal distribution. Therefore, generalizing the concept of the highroad capacity to other similar problems (e.g., a network of pipes, a communication network, etc.) the distribution (20) can be considered sufficiently generic for the numerical experiments of this section.*

Given the stochastic flow networks $\mathcal{G}_1$ and $\mathcal{G}_2$, the corresponding maximum-flow regression problems consist of the approximation of the functions $\boldsymbol{F}_1 : \mathbb{R}^n \to \mathbb{R}^m$, $n = 126$, $m = 15$, and $\boldsymbol{F}_2 : \mathbb{R}^n \to \mathbb{R}^m$, $n = 269$, $m = 15$, respectively, where $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ are defined as in (16). For each $i = 1, 2$, we build the dataset $\mathcal{D}_i$ of $\mathcal{G}_i$ made of 10,000 pairs $(\boldsymbol{c}, \boldsymbol{\varphi} = \boldsymbol{F}_i(\boldsymbol{c})) \in \mathbb{R}^n \times \mathbb{R}^m$, where 3000 of them are used as the test set ($\mathcal{P}_i$) and the remaining 7000 pairs are used to generate the multi-set Θ_i sampling $\vartheta \in \{1, \dots, 7000\}$ pairs. In particular, 80% of the pairs in Θ_i are used as the training set ($\mathcal{T}_i$) and the remaining 20% are used as the validation set ($\mathcal{V}_i$). An important aspect of our numerical experiments consists of analyzing the performance of the trained NNs, varying the quantity of available data for the training process (i.e., ϑ), and not only varying the hyper-parameters related to the architecture and optimization method; in particular, we study the NN performances when the number of training and validation data is $\vartheta = 7000, 1000, 500$. Indeed, in real-world problems, the amount of available data can be limited for many reasons (e.g., limited computational resources for simulations, limited time for measurements, etc.). Then, studying the performances of a regression model while decreasing ϑ is important to understand the sample efficiency of the model.

The fluxes $\boldsymbol{F}_i(\boldsymbol{c}) = \boldsymbol{\varphi}$ for the dataset creation are computed using the `maximum_flow` NetworkX function that, specifically, allows the computatio of the flows for all the edges of the network (given the capacities $\boldsymbol{c}$). Then, considering all the 10,000 simulations executed

to build the dataset $\mathcal{D}_i$, and denoting $\ell^{(i)}_{\max}(\boldsymbol{c})$ the length of the longest source-sink path in $\mathcal{G}_i(\boldsymbol{c})$, we observe that:

1. $\ell_{\min}(\mathcal{G}_1) = 4$, $\ell_{\text{av}}(\mathcal{G}_1) \approx 5.5$, and $\ell_{\max}(\mathcal{G}_1) = 9$;
2. $\ell_{\min}(\mathcal{G}_2) = 7$, $\ell_{\text{av}}(\mathcal{G}_2) \approx 10.7$, and $\ell_{\max}(\mathcal{G}_2) = 17$;

where $\ell_{\min}(\mathcal{G}_i)$, $\ell_{\text{av}}(\mathcal{G}_i)$, and $\ell_{\max}(\mathcal{G}_i)$ are the minimum, the average, and the maximum lengths, respectively, of the longest source-sink path of the flow for $\mathcal{G}_i$ with respect to $\mathcal{D}_i$, i.e.,

$$\ell_{\min}(\mathcal{G}_i) := \min_{(\boldsymbol{c},\boldsymbol{\varphi})\in\mathcal{D}_i} \ell^{(i)}_{\max}(\boldsymbol{c})\,, \tag{21}$$

$$\ell_{\text{av}}(\mathcal{G}_i) := \frac{1}{|\mathcal{D}_i|} \sum_{(\boldsymbol{c},\boldsymbol{\varphi})\in\mathcal{D}_i} \ell^{(i)}_{\max}(\boldsymbol{c})\,, \tag{22}$$

and

$$\ell_{\max}(\mathcal{G}_i) := \max_{(\boldsymbol{c},\boldsymbol{\varphi})\in\mathcal{D}_i} \ell^{(i)}_{\max}(\boldsymbol{c})\,. \tag{23}$$

The values reported in items 1 and 2 show that, on the average, in a radius of the length $\ell_{\text{av}}(\mathcal{G}_i)$ from the sink t, it is likely to find almost all the nodes characterizing the maximum-flow of the network $\mathcal{G}_i$. This information is then taken into account while choosing the depth values for the construction of the GINNs in the following Section 3.3.1. Indeed, we recall that the number of consecutive GI layers in an NN tells us if the input feature of node v_i contributes to the computation of the output feature of node v_j (see Proposition 1). Therefore, it is interesting to verify if the regression performance of a GINN improves or not, when the number of GI layers is related to one of the quantities (21)–(23).

3.3.1. NN Architectures, Hyper-Parameters, and Training

In the numerical experiments of this section, we study and compare the performances of MLPs and GINNs concerning the maximum-flow regression problems related to $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$. Then, we consider the two archetypes of NN architectures: an MLP archetype and a GINN archetype.

- **MLP Archetype:** The NN architecture is characterized by one input layer L_0, $H \in \mathbb{N}$, hidden layers $L_1, \ldots, L_H$ with a nonlinear activation function f, and one output layer L_{H+1} with a linear activation function. The output layer is characterized by m units, while all the other layers are characterized by n units. Finally, we apply a batch normalization [43] before the activation function for each hidden layer $L_1, \ldots, L_H$. See Figure 4.
- **GINN Archetype:** The NN architecture is characterized by one input layer L_0 of n units, $H \in \mathbb{N}$ hidden GI layers $L^{GI}_1, \ldots, L^{GI}_H$ with a nonlinear activation function f, and one output layer L^{GI}_{H+1} with a linear activation function. All the GI layers are built with respect to the adjacency matrix $A_L \in \mathbb{R}^{n\times n}$ of the line graph of the network (see Section 3.2.1) and they are characterized by $F \in \mathbb{N}$ filters (i.e., output features). Then, the number of input features K of the GI layer L^{GI}_h is $K = F$, if $h > 1$, and $K = 1$, if $h = 1$. As for the MLP archetype, we apply a batch normalization before the activation function of each hidden layer. Finally, the output layer is characterized by a pooling operation and by the application of a mask (see Section 2.3) to focus on the m units corresponding to the m target flows. See Figure 5.

For our experiments, given the two NN archetypes above, we built a set of untrained NN models, varying the main hyper-parameters of the architectures. In particular, for the MLPs, we varied the hyper-parameters H and f (i.e., the depth and activation functions of the hidden layers), while for the GINNs, we also varied F (i.e., the number of filters of the GI layers) and the pooling operation. Specifically, the hyper-parameters vary among these values:

- **MLP archetype.** $f \in \{\text{relu}, \text{elu}, \text{swish}, \text{softplus}\}$ and $H \in \{2, 3, 4, 5\}$. We do not use deeper MLPs to avoid the so-called degradation problem [44], i.e., the problem in

which increasing the number of hidden layers causes the performance of an NN to saturate and degrade rapidly.

- **GINN archetype.** $f \in \{\text{relu, elu, swish, softplus}\}$, $F \in \{1, 5, 10\}$, and pooling operations in $\{\max, \text{mean}\}$ (only if $F = 5, 10$); $H \in \{3, 5, 7, 9\}$ for $\mathcal{G}_1$ and $H \in \{4, 9, 14, 19\}$ for $\mathcal{G}_2$. In particular, we select these values of H because they are a discrete interval around the value $\ell_{\text{av}}(\mathcal{G}_i)$, also including cases near, or equal to, the minimum and maximum values $\ell_{\min}(\mathcal{G}_i)$ and $\ell_{\max}(\mathcal{G}_i)$, respectively.

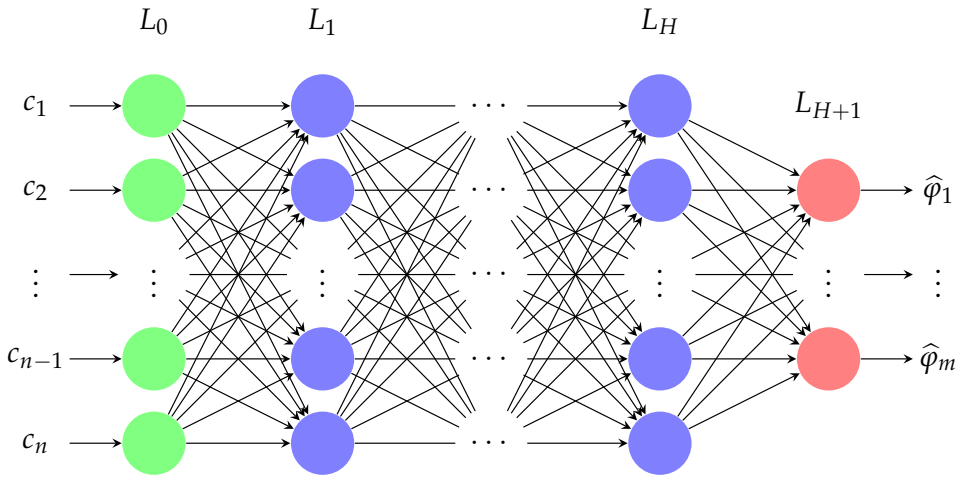

Figure 4. MLP archetype. The units of the input layer L_0 are in green, the units of the hidden layers $L_1, \ldots, L_H$ are in purple, and the units of the output layer L_{H+1} are in red.

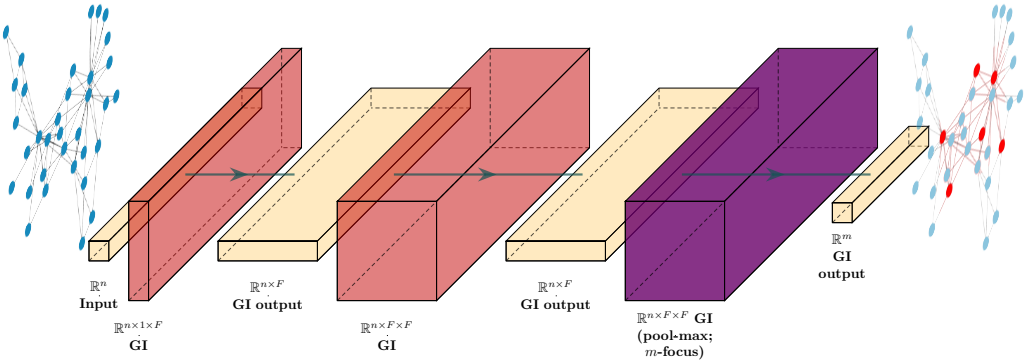

Figure 5. Example of a GINN archetype with depth $H = 2$ and max-pooling operation for the output layer. The output matrices Y of the NN layers are in orange, the weight tensors $\mathbf{W}$ of the hidden GI layers are in red (see Definition 3), and the weight tensor $\mathbf{W}$ of the output GI layer with max-pooling and masking operations (see Section 2.3) are in purple.

Then, these models are all trained on $\vartheta = 7000, 1000, 500$ input–output pairs sampled from $\mathcal{D}_i - \mathcal{P}_i$, using a mini-batch size $\beta = 128, 64, 32$; the weight initialization is a Glorot normal distribution [45] for the MLPs and it varies among a Glorot normal and a normal distribution $\mathcal{N}(0, 0.5)$ for the GINNs. All the biases are initialized as zeroes.

The remaining training options are fixed and shared by all the models during the training. In particular, these options are:

- Mean square error (MSE) loss, i.e.,

$$\text{loss}(\mathcal{B}) := \frac{1}{m} \sum_{j=1}^{m} \left(\frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{c},\boldsymbol{\varphi})\in\mathcal{B}} (\widehat{\varphi}_j - \varphi_j)^2 \right), \tag{24}$$

where $\mathcal{B}$ is any generic batch of input–output pairs;

- The Adam optimizer [46] (learning rate $\epsilon = 0.002$, moment decay rates $\rho_1 = 0.9$, $\rho_2 = 0.999$);
- Early stopping regularization [20,47] (200 epochs of patience, restore best-weights), to avoid overfitting;
- Learning rate reduction on plateau [47] (reduction factor $\alpha = 0.5$, 100 epochs of patience, minimum learning rate $\epsilon = 10^{-6}$).

The training of all the NN models, with respect to all the different training configurations, returns 3168 trained NNs; in particular, we have 144 MLPs and 1140 GINNs, for each stochastic flow network $\mathcal{G}_1$ and $\mathcal{G}_2$.

3.3.2. Performance Analysis of Maximum-Flow Regression

To evaluate the performance of an NN trained with respect to the maximum-flow regression task, we consider the errors MRE_{av} and MRE_{φ} (see Section 3.2 and Equations (18) and (19), respectively) measured on the test set. In particular, to better analyze the performances, we visualize the NNs as points in the $(\text{MRE}_{av}, \text{MRE}_{\varphi})$ plane (see Figures 6 and 7). Then, the nearer a point is to the origin (i.e., the ideal zero-error NN), the better the regression performances of the corresponding NNs are. We decide to use this representation because of the characteristics of the errors reported in Remark 6. Indeed, it is important to analyze the behavior of the NNs with respect to MRE_{av} and MRE_{φ} together.

We start the analysis with the first stochastic flow network $\mathcal{G}_1$. In general, looking at Figure 6, we clearly see that the GINNs have better regression performances than the MLPs. In particular:

1. The MRE_{φ} of the GINNs is generally smaller than the MLPs, and this effect increases with ϑ;
2. The MRE_{av} of the GINNs is almost always smaller than the MLPs, and this effect seems to be almost stable while varying ϑ;
3. Looking at the hyper-parameter F, we observe that the cases with $F = 1, 10$ generally perform better with fewer training samples (i.e., $\vartheta = 1000, 500$) while the cases with $F = 5$ generally perform better with $\vartheta = 7000$. This phenomenon suggests that increasing the number of filters can improve the quality of the training, even if a clear rule for the best choice of F is not apparent.

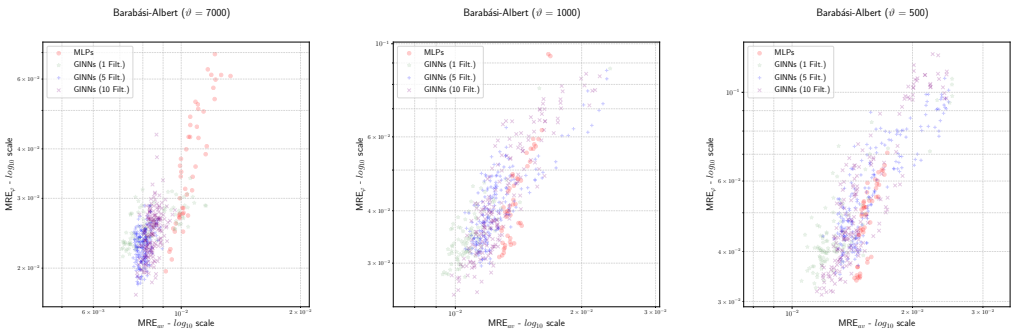

Figure 6. Network $\mathcal{G}_1$. Scatter plots in the $(\text{MRE}_{av}, \text{MRE}_{\varphi})$ plane. Left to right: NNs trained with $\vartheta = 7000, 1000, 500$ samples. Red circles: MLPs; green stars, blue crosses and purple "x": GINNs with $F = 1, 5, 10$, respectively.

We continue the analysis with the second stochastic flow network $\mathcal{G}_2$, increasing the size and complexity of the flow network. Indeed, the graph G_{BA} of $\mathcal{G}_1$ is characterized by a reduced complexity of the maximum-flow problem, because the BA graphs are generated using a preferential attachment criterion that keeps the average length of the maximum source-sink path $\ell_{\text{av}}(\mathcal{G}_1)$ small, even when increasing the nodes of the graph (this phenomenon was observed during some preliminary experiments).

Looking at Figure 7, we notice the same characteristics observed for $\mathcal{G}_1$, but much more emphasized. In particular, for each $\vartheta = 7000, 1000, 500$, we clearly see that the GINNs

generally outperform the MLPs, especially with respect to the MRE_{av}. The reason for these similarities probably lies in the nature of the graphs G_{BA} and G_{ER} of $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively; indeed, the ER graphs are used to represent generic graphs and are typically used to show properties that hold for almost all the graphs. On the other hand, the graph G_{BA} is simpler than G_{ER}. Then, it is reasonable that the observations made for $\mathcal{G}_1$ are confirmed looking at $\mathcal{G}_2$ and it is reasonable that the performance differences observed in $\mathcal{G}_2$ are less emphasized in $\mathcal{G}_1$, since the maximum-flow problem on G_{BA} is less complex than on G_{ER}.

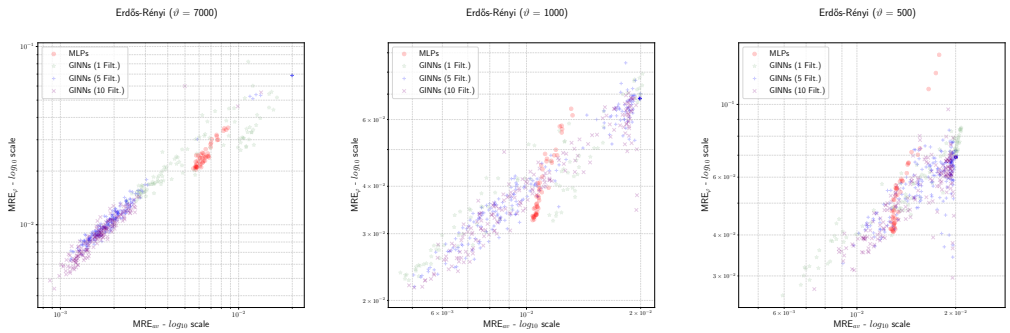

Figure 7. Network $\mathcal{G}_2$. Scatter plots in the $(\text{MRE}_{av}, \text{MRE}_{\varphi})$ plane. Left to right: NNs trained with $\vartheta = 7000, 1000, 500$ samples. Red circles: MLPs; green stars, blue crosses and purple "x": GINNs with $F = 1, 5, 10$, respectively.

Remark 8 (GINNs, small MRE_{av}, and regressions on graphs)**.** *We have just observed that the GINNs genrally perform better than MLPs for regression tasks on graphs but, if we focus on the MRE_{av} values, the GINNs clearly show better performances (see Figures 6 and 7 and Tables 1 and 2). Specifically, Tables 1 and 2 show the three GINNs and MLPs with lowest MRE_{av} value on the test sets of $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. The better performances of the GINNs, with respect to this error measure, are particularly important if we extend the regression problem, e.g., if we want to learn all the flow values $\varphi^{*(c)}(e_1), \ldots, \varphi^{*(c)}(e_n)$ on the edges of the graph and not only the ones characterizing the maximum-flow φ reaching the sink. Indeed, in this case, an NN with small errors on the single elements of the target vector is fundamental, while an NN with small errors on the sum of the elements of the target vector is useless; for this reason, in vector-valued regression tasks, we use loss functions such as (24) (evidently similar to the performance measure MRE_{av}). In conclusion, we believe that the GINNs have great potential in the field of regression graphs.*

We continue to conclude the performance analysis, and we analyze how the errors of the NN models change when varying the hyper-parameters.

The first observation is related to the activation functions and the mini-batch size. We observe that the trained NN models (both GINNs and MLPs) generally exhibit worse regression performances with a relu activation function and a mini-batch size equal to 128; in Figure 8, we report the same scatterplots of Figures 6 and 7 but without the points corresponding to the NNs with the relu activation function or a mini-batch size equal to 128. It is worth noting that the general observations concerning the NN performances are even more evident when removing these models. Moreover, among the remaining models, we do not observe activation functions or mini-batch size values that are evidently better than the others; in general, as can be expected, we only observe a few advantages of using a mini-batch size of 32 samples instead of a mini-batch size of 64 samples, while decreasing ϑ.

From now on, we do not include in our analyses the models characterized by a relu activation function or mini-batch size equal to 128. Moreover, we only focus on the GINN models and, in particular, on their performances with respect to the hyper-parameter H, characterizing the number of hidden layers. Indeed, the remaining hyper-parameters (pooling operations and weight initializations) do not seem to have a particular impact on the results.

The study of the GINN performances with respect to H is particularly interesting if we consider Proposition 1. In fact, from this proposition, we expect the GINN models to

have a better performance if the depth H is such that $H+1 \gtrsim \ell_{\text{av}}(\mathcal{G}_i)$. This guess is indeed satisfied. In particular, for $\mathcal{G}_1$, we see that by increasing the depth H, we obtain GINNs with better performances in general (see Figure 9). On the other hand, for $\mathcal{G}_2$ we observe a slightly different behavior. The GINNs that are sufficiently deep (i.e., $H \geq 9$) show better performances than the GINNs with $H = 4$, but their errors tend to increase with a small ϑ; in particular, the more H is greater than $\ell_{\text{av}}(\mathcal{G}_2)$, the more the GINN performances seem to deteriorate (see Figure 10). To summarize, the depth in a GINN model is very important to obtaining good regression abilities, keeping in mind Proposition 1. Nonetheless, the practice of using as much of a GI layer as possible is not always the best choice, and this topic deserves attention in future work.

Table 1. Network $\mathcal{G}_1$. Top three GINNs and MLPs, for $\vartheta = 7000, 1000, 500$. Models are sorted with respect to the MRE_{av} error; the "rank" column describes their global position with respect to all the other models.

$\mathcal{G}_1$	GINNs								MLPs				
ϑ	**Rank**	MRE_{av}	H	F	f	β	**Pool.**	**Init.**	**Rank**	MRE_{av}	H	f	β
7000	1/528	0.00707	9	1	elu	64	-	G.Norm.	446/528	0.00914	3	swish	32
	2/528	0.00712	9	1	elu	128	-	Norm.	453/528	0.00934	3	swish	64
	3/528	0.00713	9	1	elu	32	-	Norm	455/528	0.00939	4	swish	32
1000	1/528	0.00933	9	1	softplus	32	-	G.Norm.	338/528	0.01272	5	softplus	32
	2/528	0.00940	7	1	elu	32	-	Norm.	353/528	0.01289	4	elu	32
	3/528	0.00952	7	1	elu	32	-	G.Norm.	357/528	0.01292	5	elu	32
500	1/528	0.01056	7	1	softplus	32	-	Norm.	263/528	0.01433	5	softplus	32
	2/528	0.01077	5	1	relu	32	-	G.Norm.	279/528	0.01448	5	softplus	64
	3/528	0.01092	7	1	softplus	32	-	G.Norm.	284/528	0.01452	4	softplus	32

Table 2. Network $\mathcal{G}_2$. Top three GINNs and MLPs, for $\vartheta = 7000, 1000, 500$. Models are sorted with respect to the MRE_{av} error; the "rank" column describes their global position with respect to all the other models.

$\mathcal{G}_1$	GINNs								MLPs				
ϑ	**Rank**	MRE_{av}	H	F	f	β	**Pool.**	**Init.**	**Rank**	MRE_{av}	H	f	β
7000	1/528	0.00087	14	10	softplus	32	max	G.Norm.	442/528	0.00557	2	elu	32
	2/528	0.00092	14	10	softplus	32	mean	G.Norm.	445/528	0.00571	5	softplus	64
	3/528	0.00099	12	10	softplus	32	mean	G.Norm	446/528	0.00573	5	elu	32
1000	1/528	0.00465	9	1	elu	32	-	G.Norm.	263/528	0.01038	3	softplus	32
	2/528	0.00478	19	1	elu	32	-	G.Norm.	264/528	0.01038	2	swish	32
	3/528	0.00479	14	1	softplus	32	-	Norm.	267/528	0.01043	3	softplus	64
500	1/528	0.00593	14	1	elu	32	-	G.Norm.	114/528	0.01266	2	swish	32
	2/528	0.00674	14	1	softplus	32	-	G.Norm.	118/528	0.01272	2	swish	64
	3/528	0.00688	19	1	softplus	32	-	G.Norm.	119/528	0.01272	5	softplus	32

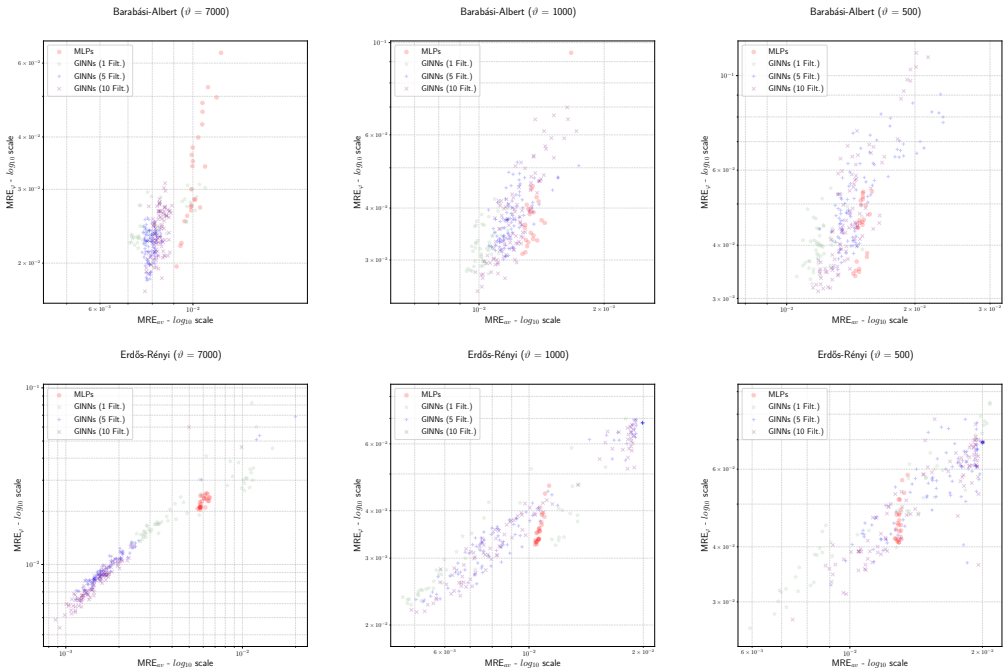

Figure 8. Scatter plots in the $(\mathrm{MRE}_{av}, \mathrm{MRE}_{\varphi})$ plane for NNs trained with respect to $\mathcal{G}_1$ (**top**) and $\mathcal{G}_2$ (**bottom**). Left to right: NNs trained with $\vartheta = 7000, 1000, 500$ samples. Red circles: MLPs; green stars, blue crosses and purple "x": GINNs with $F = 1, 5, 10$, respectively. We do not plot results with NNs that have a relu activation function or a mini-batch size equal to 128.

Remark 9 (Training time)**.** *Along the conducted experiments (with training that occurred more than 3000 times), we find that the average training time for the GINN models is approximately 20 min in total, and one second per epoch; on the other hand, the average training time for the MLP models is approximately 10 min in total and half a second per epoch. Nonetheless, we point out that the difference in training times between GINNs and MLPs can be reduced with code optimization. Indeed, the GINN layers are a custom class of TensorFlow NN layers developed on purpose by the authors for numerical experiments. The code of TensorFlow's FC layers is extremely optimized. Therefore, at the present time, the GINNs and MLPs cannot be compared in terms of computational costs. More details about the average training times per epoch of the models are reported in Table 3, and this quantity is indicative of the training computational cost of the NNs. However, we recall that the experiments take into account more than three thousand models, each one with a different training configuration that characterizes the training time per epoch. All the training was performed on a workstation with a CPU of 4 Core and 8 Threads, 32 GB of RAM, and a GPU Nvidia 1080 8 GB.*

Table 3. Global statistics of the average training time per epoch for GINN and MLP models, expressed in seconds.

	Avg. Time per Epoch (s)	
	GINNs	**MLPs**
Mean	1.099	0.565
Std	1.359	0.292
25th perc.	0.318	0.380
50th perc.	0.567	0.431
75th perc.	1.296	0.632

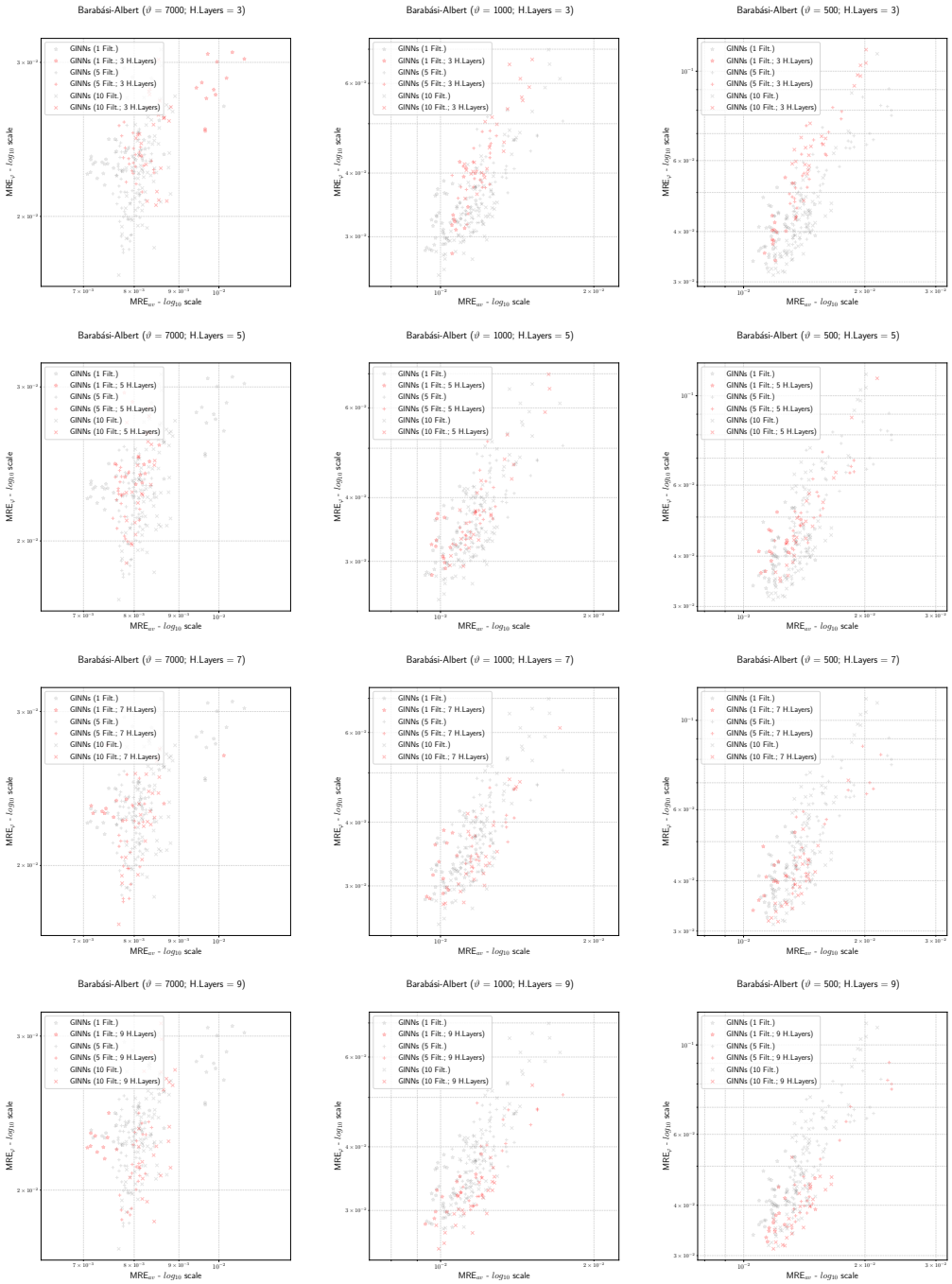

Figure 9. Scatter plots in the $(\mathrm{MRE}_{av}, \mathrm{MRE}_{\varphi})$ plane for GINNs trained with respect to $\mathcal{G}_1$. Left to right: GINNs trained with $\vartheta = 7000, 1000, 500$ samples; top to bottom: red markers highlight GINNs with $H = 3, 5, 7, 9$ (black markers for all the other models).

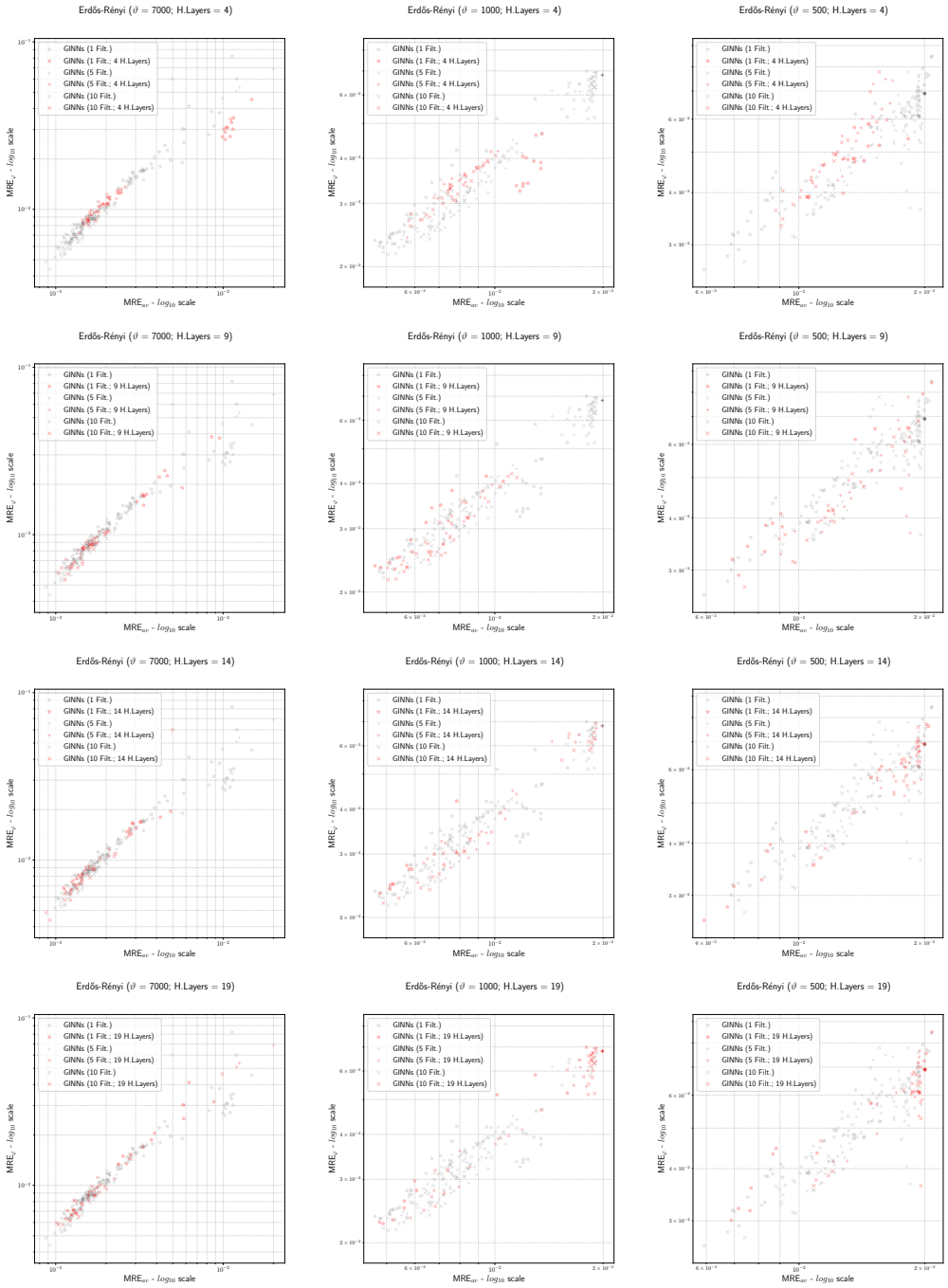

Figure 10. Scatter plots in the $(\mathrm{MRE}_{av}, \mathrm{MRE}_{\varphi})$ plane of the GINNs trained with respect to $\mathcal{G}_2$. From left to right, the GINNs are trained using $\vartheta = 7000, 1000, 500$ samples; from top to bottom, the red markers highlight the GINNs with hyper-parameters of $H = 4, 9, 14, 19$ (black markers for all the other models).

3.4. GINNs for Flux Regression in Discrete Fracture Networks

In Section 3.3, we showed the regression abilities and the potentialities of the GINN models for the maximum-flow regression problem, i.e., for a problem representative of generic real-world applications.

In this section, we address a specific real-world application where GINNs can be useful; in particular, we consider an uncertainty quantification (UQ) problem related to underground flows in fractured media. Flow characterization in underground fractured media is an interesting problem for many applications, such as civil engineering, industrial engineering, and environmental analyses. A helpful model to describe the flow in an

underground network of fractures is represented by the discrete fracture network (DFN) models [48–50]. These models express the fractures as two-dimensional polygons into a three-dimensional domain, and each fracture is described by specific hydro-geological properties (e.g., fracture transmissivity) and geometrical properties (e.g., barycenter position and orientation). Intersections between fractures, denominated "traces", define flux exchange phenomenons, such that the flow model is typically ruled by the following assumptions: (i) the rock matrix is impenetrable; (ii) the Darcy law is used for the flux propagation on each fracture; and (iii) the head continuity and flux balance is imposed on all the traces. However, since the hydro-geological and geometrical characteristics of the network of fractures are usually not accessible in detail, the DFN models are typically generated by sampling their hydro-geological and geometrical features from known distributions [51–54]. Therefore, a statistical approach is required to study real fractured media, resorting to UQ analyses.

In recent literature, several new methods have been proposed to reduce the computational costs of the DFN flow simulations (e.g., see [55–57]); nonetheless, they are still computationally expensive in many situations and the UQ analyses can involve thousands of these simulations. Therefore, it is fundamental to take into account the techniques for the complexity reduction, such as machine learning-based techniques (e.g., see [16,17,58]). In particular, in [16,17], NN models are trained on datasets built using DFN simulations to provide surrogate models; finally, in a negligible amount of time, the NN models are used to generate a large set of approximated DFN flow simulation results, which are particularly useful to speed up the UQ analyses.

In this section, the idea is to take advantage of the DFN's graph structure to build GINN models and to analyze the advantages of using such models for the DFN flux regression tasks instead of the more classic NN architectures (e.g., MLPs or multi-task NNs).

3.4.1. The DFN Model and the Flux-Regression Task in DFNs

Here, for the reader's convenience, we briefly describe the problem of the flow simulations in DFNs. We point the interested reader to [55–57] for full details.

A DFN model is a discrete representation of an underground network of fractures in a fractured rock medium using a set of intersecting planar polygons (the fractures) in a three-dimensional domain $\mathbb{D} \subset \mathbb{R}^3$ (the rock medium); see Figure 11 for a DFN example. Each fracture (i.e., polygon) is labeled and identified by an index belonging to an arbitrary set I; i.e., for each $i \in I$, we denote each fracture by $\mathcal{F}_i$. Then, a DFN is defined as the union of all the fractures, i.e., $\cup_{i \in I} \mathcal{F}_i$. Since the traces, i.e., the segments obtained from the intersection of two or more fractures that connect the fractures define a network, a DFN can be represented as a graph where the fractures are the nodes and the traces are edges (see Figure 11).

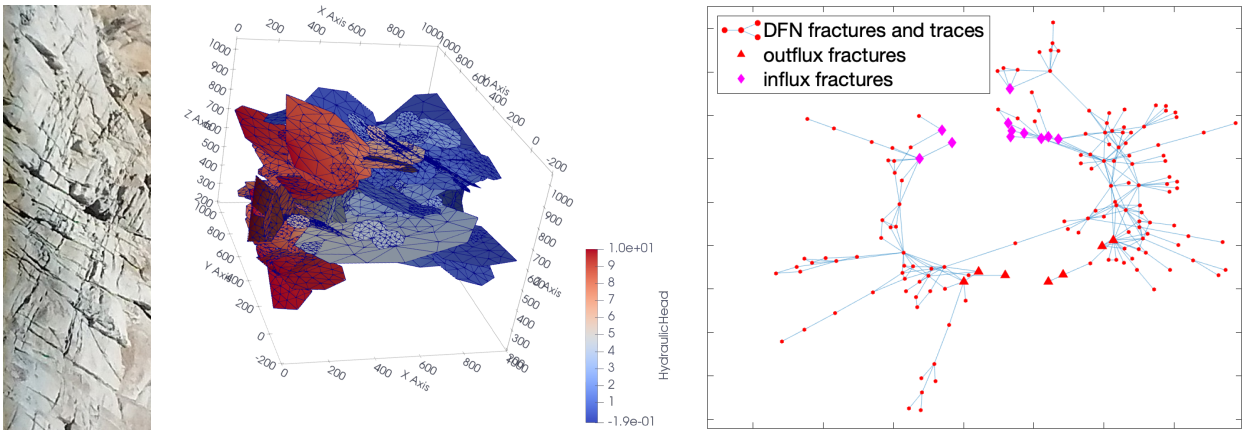

Figure 11. External surface of a natural fractured medium (**left**), a 3D view of a DFN (**center**), and a graph representation of the same DFN (**right**). The DFN illustrated in this figure is DFN158, i.e., the one used for the numerical tests of Section 3.4.

Each fracture in a DFN is characterized by a transmissivity parameter κ_i and by geometrical properties, such as the size and the orientation in $\mathbb{R}^3$. These characteristics are generally sampled from known probability distributions, and they characterize the flow simulations of the DFN. Indeed, a DFN flow simulation depends both on the geometry and the hydro-geological properties of the fractures, such as the transmissivities κ_i of the fractures.

The DFN flux regression problem addressed in this section is characterized as follows. The DFN considered consists of $n = 158$ fractures and we donote it by DFN158. The geometry of DFN158 is fixed, and the fractures are immersed into a domain $\mathbb{D}$ described by a cube of 1000 m along the long edge (see Figure 11). The domain boundary conditions are such that a fixed pressure difference $\Delta H = 10$ m is imposed between two opposite faces of $\mathbb{D}$, therefore representing an inlet and outlet face, respectively. Moreover, those fractures cut by the inlet and outlet faces of the domain are defined as inflow and outflow fractures, respectively. The edges of all the other fractures are insulated (homogeneous Neumann condition). It is possible to observe that the fixed geometry and the given boundary conditions affect the flux directionality in DFN158; on the other hand, the fracture transmissivities characterize the flow intensity exiting from the outflow fractures. In particular, we assume that the fracture transmissivities of the fractures $\mathcal{F}_1, \ldots, \mathcal{F}_n$ are isotropic parameters $\kappa_1, \ldots, \kappa_n$, respectively, described by random variables with a log-normal distribution [51,52]:

$$\log_{10} \kappa_i \sim \mathcal{N}(-5, 1/3), \quad \forall\, i = 1, \ldots, n\,.$$

In the test case considered, we use octagons to represent the fractures and they have been randomly created according to the distributions in [53,54]. For their geometrical properties: a truncated power law distribution for the fracuture radii, with an exponent $\gamma = 2.5$ and an upper and lower cut-off $r_u = 560$ and $r_0 = 50$, respectively; a Fischer distribution for fracture orientations, with a mean direction $\boldsymbol{\mu} = (0.0065, -0.0162, 0.9998)$ and a dispersion parameter of 17.8; and a uniform distribution for the mass centers of the fractures. However, the first eight fractures $\mathcal{F}_1, \ldots, \mathcal{F}_8$ are not randomly sampled but are created to ensure an inlet-outlet path for the flow, where $\mathcal{F}_1$ is an inflow fracture and $\mathcal{F}_8$ is an outflow fracture.

Together with the boundary conditions described above, the geometry of DFN158 is such that we have $m = 7$ outflow fractures in DFN158. Therefore, we can define a flux function representing the DFN flow simulations for DFN158; i.e., a function $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ such that, for each vector $\boldsymbol{\kappa} = [\kappa_1, \ldots, \kappa_n]^\top \in \mathbb{R}^n$ of fracture transmissivities with $n = 158$, it returns the vector $\boldsymbol{F}(\boldsymbol{\kappa}) = \boldsymbol{\varphi} = [\varphi_1, \ldots, \varphi_m]^\top \in \mathbb{R}^m$ of fluxes exiting from the m outflow fractures, $m = 7$.

Now, assuming the need to perform UQ analyses of the fluxes $\varphi_1, \ldots, \varphi_m$ of DFN158 while varying the fracture transmissivities, it is worth considering the flux regression problem that looks for NN-based approximations $\widehat{\boldsymbol{F}}$ of $\boldsymbol{F}$.

3.4.2. Performance Analysis of DFN Flux Regression

In this subsection, we extend the numerical experiments and analyses of Section 3.3 to the example represented by DFN158. Then, we study and compare the performances of the MLPs and GINNs concerning the flux regression problem related to the function $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ described in Section 3.4.1. The two archetypes of MLP and GINN architectures are the same as in Section 3.3.1, as are most of the hyper-parameter values and the training options used; the only differences are the following:

- $\vartheta = 1000, 500$ (number of training and validation data);
- $\beta = 64, 32$ (mini-batch size);
- The relu activation function is not considered in the experiments;
- For the GINN models, we consider the depth parameter values $H \in \{4, 7, 9, 14, 19\}$. The rationale behind this choice is that it is a set of values around 8, which is the number of deterministic fractures that, on average, represent an inlet-outlet flow path

for DFN158 (in the absence of a value equivalent to ℓ_{av} that cannot be easily computed for DFN158);

- The GI layers are built with respect to the adjacency matrix A of DFN158; indeed, we do not need to introduce the line graph of the network since the features (i.e., the transmissivities) are assigned to the nodes of the graph and not to the edges.

As in Section 3.3.2, we evaluate the performance of the NNs trained with respect to the DFN flux regression task measuring the errors MRE_{av} and MRE_{φ} on the test set (also, in this case, $\mathcal{P}$ is made of 3000 samples). Then, we visualize the results of the NNs as points in the $(\text{MRE}_{av}, \text{MRE}_{\varphi})$ plane (see Figure 12). Analyzing the error values and looking at the scatter plots of Figure 12, we clearly observe that the GINN models outperform the MLPs, and that they are characterized by more regular error behaviors than the GINNs trained for the maximum-flow regression task, with respect to the filter hyper-parameter (see Section 3.3.2). In particular:

1. Both the MRE_{φ} and the MRE_{av} of the GINNs are almost always smaller than the ones of the MLPs, independently of ϑ;
2. Looking at the filter hyper-parameter F, we observe that the GINN performances are better as F increases (from $F = 1$, to $F = 5$, to $F = 10$).

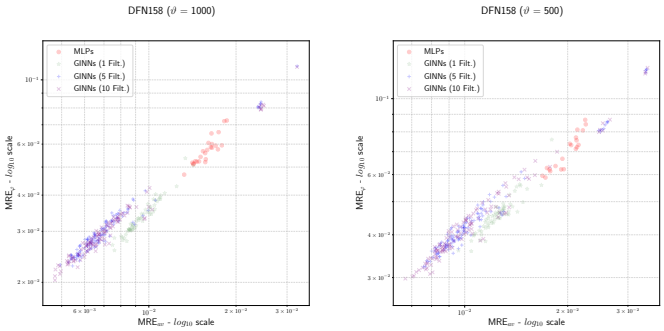

Figure 12. Scatter plots in the $(\text{MRE}_{av}, \text{MRE}_{\varphi})$ plane for NNs trained with respect to DFN158. NNs are trained using $\vartheta = 1000$ (**left**) and $\vartheta = 500$ (**right**) samples. Red circles: MLPs; green stars, blue crosses and purple "x": GINNs with $F = 1, 5, 10$, respectively.

We continue the analysis of studying the relationships between the GINN errors and the other hyper-parameters of the models. With respect to the activation functions, we observe that the GINN models with the elu activation functions have, in general, slightly better performances than other models; on the other hand, all the GINN models with the worst performances (i.e., the points in the top-right corners of Figure 12) have softplus activation functions. Concerning the mini-batch size β, we observe that the GINNs with the best performances (corresponding to points in the bottom-left corners of Figure 12) are trained with $\beta = 32$. Similar results hold for the weights initialization, where the best performing GINNs are initialized with a Glorot normal distribution. Concerning the pooling operations, we do not observe particular differences in the error values of GINN models using a max-pooling or a mean-pooling operation.

Analogous to Section 3.3.2, we conclude with a focus on the error behaviors with respect to the depth H of the GINN models. Moreover, for the DFN flux regression task, we observe that the depth H of the model can improve the regression quality. In particular, for each $\vartheta = 1\,000, 500$, we observe that the best performances are obtained by the GINNs with a depth of $H = 7, 9, 14$, while both the shallowest and the deepest GINNs ($H = 4, 19$) have higher errors (see Figures 13 and 14). In accordance with Proposition 1 and the observations of Section 3.3.2, this characteristic let us deduce that, on average, the maximum inlet-outlet flux path in DFN158 is probably made of 8 to 15 fractures; i.e., a value not so far from the length of the inlet-outlet path defined by the fractures $\mathcal{F}_1, \ldots, \mathcal{F}_8$.

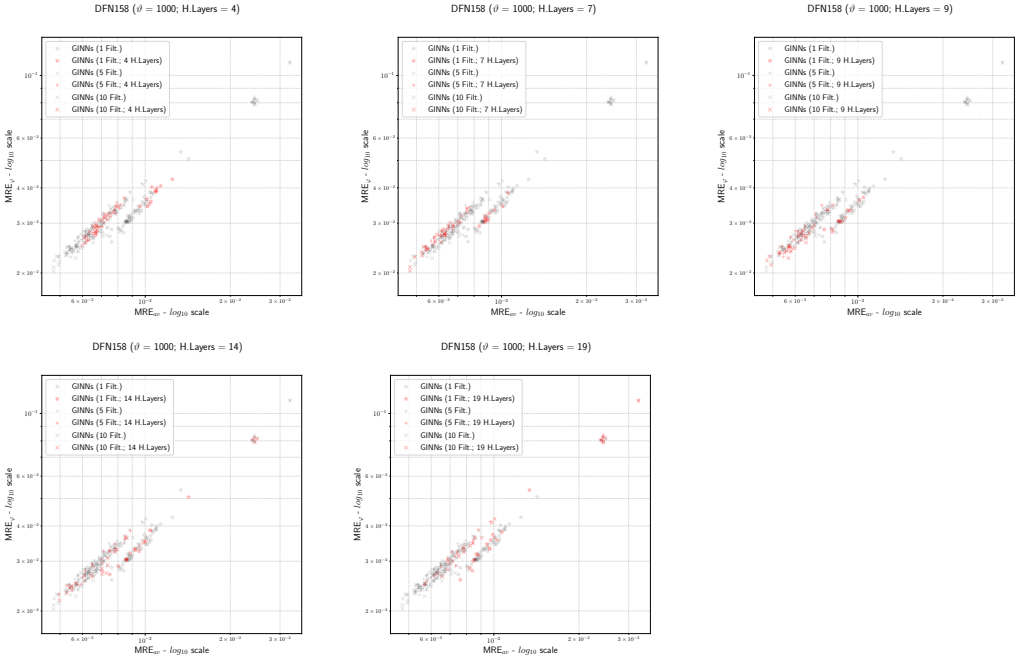

Figure 13. Scatter plots in the $(\mathrm{MRE}_{av}, \mathrm{MRE}_{\varphi})$ plane for GINNs trained with respect to DFN158, $\vartheta = 1000$. Left to right, top to bottom: red markers highlight GINNs trained with $H = 4, 7, 9, 14, 19$, respectively (black markers for all the other models).

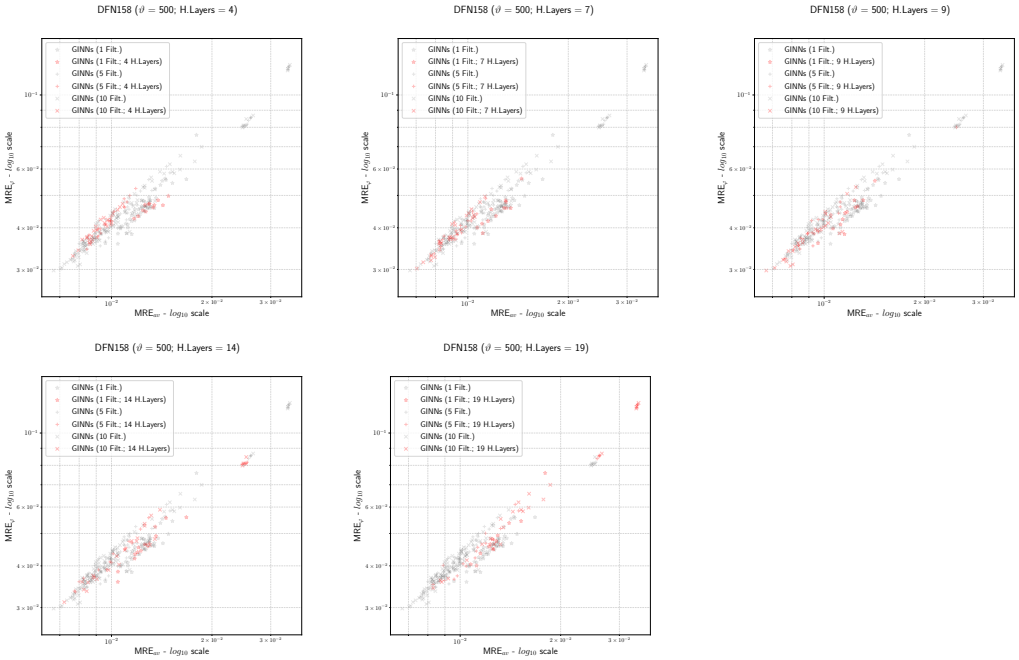

Figure 14. Scatter plots in the $(\mathrm{MRE}_{av}, \mathrm{MRE}_{\varphi})$ plane for GINNs trained with respect to DFN158, $\vartheta = 500$. Left to right, top to bottom: red markers highlight GINNs trained with $H = 4, 7, 9, 14, 19$, respectively (black markers for all the other models).

4. Conclusions

In this work, we presented the graph-informed (GI) layers, a new type of spatial-based graph convolutional layer, designed for regression tasks on graph-structured data. With respect to the other types of GCN layers, our GI layers stand out for their tensor formulation (see (8)), managing multiple input/output features; moreover, these layers let users build deep NN architectures which are able to exploit the depth needed to improve

the regression performances (see Proposition 1 and Sections 3.3.2 and 3.4.2). The GI layers have been formally defined, from the simplest version to the most general and tensor version. Moreover, additional optional operations are introduced: the pooling operation and mask operations.

To study the regression abilities of graph-informed NNs (GINNs), i.e., NNs made from GI layers, we trained thousands of NN models (both GINNs and MLPs) on two maximum-flow regression problems, with networks based on a Barabási–Albert graph ($\mathcal{G}_1$) and an Erdős–Rényi graph ($\mathcal{G}_2$). We selected the maximum-flow regression problem as a representative test since it is a sufficiently general problem to demonstrate the applications in many topics of the network analysis. Analyzing the approximation errors of the NNs, we observed that the GINNs have better performances than the MLPs. in general. In particular, for $\mathcal{G}_2$, the GINNs in almost all the cases outperform the MLPs. The study of the regression performances also showed an interesting relationship between small errors and a depth greater than, or equal to, the average length of the maximum source-sink path in the stochastic network.

After the test on the maximum-flow regression task, we illustrated an example of a possible application of the GINNs to a real-world problem: a DFN flux regression problem, i.e., an uncertainty quantification problem for the characterization of the exiting flux distribution of an underground network of fractures. In this practical application, the GINN models completely outperform the MLPs; moreover, both the depth and the filter hyper-parameters of the GINNs proved to be significant enough to improve the approximation quality of the target function.

In conclusion, we believe that our work introduces a new, useful, contribution to the family of spatial-based graph convolutional networks; indeed, the numerical experiments illustrated here show that the new GI layers and the GINNs have great potentialities in the framework of regression tasks on graph-structured data.

Author Contributions: Conceptualization, S.B., F.D.S., A.M., S.P., F.V.; data curation, S.B., F.D.S., A.M., S.P., F.V.; formal analysis, S.B., F.D.S., A.M., S.P., F.V.; funding acquisition, S.B., S.P., F.V.; investigation, S.B., F.D.S., A.M., S.P., F.V.; methodology, S.B., F.D.S., A.M., S.P., F.V.; project administration, S.B., F.D.S., A.M., S.P., F.V.; resources, S.B., F.D.S., A.M., S.P., F.V.; software, F.D.S., A.M.; supervision, S.B., S.P., F.V.; validation, S.B., F.D.S., A.M., S.P., F.V.; visualization, F.D.S., A.M.; writing—original draft, S.B., F.D.S., A.M., S.P., F.V.; writing—review and editing, S.B., F.D.S., A.M., S.P., F.V. All authors have read and agreed to the published version of the manuscript.

Funding: Research performed in the framework of the Italian MIUR Award "Dipartimento di Eccellenza 2018–2022" to the Department of Mathematical Sciences, Politecnico di Torino, CUP: E11G18000350001. F.D.S. and S.P. also acknowledge support from Italian MIUR PRIN project 201752HKH8_003. A.M. gratefully acknowledges the support from Addfor Industriale. The research leading to these results was also partially funded by the SmartData@PoliTO center for Big Data and Machine Learning technologies.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The code used to implement the Graph-Informed Neural Networks is available at https://github.com/Fra0013To/GINN (accessed on 31 January 2022).

Acknowledgments: The authors acknowledge support from the GEOSCORE group (https://areeweb.polito.it/geoscore/, accessed on 1 February 2022) of Politecnico di Torino (Department of Mathematical Sciences).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations and nomenclatures are used in this manuscript:

BA	Barabási–Albert
CNN	Convolutional Neural Network
CwD	Circulation with Demand
DCNN	Diffusion-Convolutional Neural Networks
DFN	Discrete Fracture Network
DL	Deep Learning
ER	Erdős–Rényi
FC	Fully-Connected
GCN	Graph Convolutional Network
GI	Graph-Informed
GINN	Graph-Informed Neural Network
GNN	Graph Neural Network
ML	Machine Learning
MLP	Multi-Layer Perceptron
MRE	Mean Relative Error
MRE_{av}	Edge-Wise Average MRE
MRE_{φ}	MRE on the Predicted Maxflow/Outflow
MSE	Mean Square Error
NIM	Network Interdiction Model
NN	Neural Network
UQ	Uncertainty Quantification

Appendix A. Multi-Sets

Definition A1 (Multi-set [59,60])**.** *A multi-set $\mathcal{A}$ is a collection of objects, called elements, which may occur more than once. The number of times an element occurs in a multi-set is called its multiplicity. The cardinality of a multi-set (denoted by $|\mathcal{A}|$) is the sum of the multiplicities of its elements.*

In other words, a multi-set may be formally defined as a pair $\mathcal{A} = (A, m)$, where A is the underlying set of the multi-set, formed from its distinct elements, and $m : A \to \mathbb{Z}^+$ is a function that, for each $a \in A$, returns the multiplicity $m(a) \geq 1$ of a in the multi-set.

Definition A2 (Relations and Operations with Multi-sets [60])**.** *The usual relations and operations on sets can be extended to multi-sets by considering the multiplicity function. Let $\mathcal{A} = (A, m_A)$ and $\mathcal{B} = (B, m_B)$ be two multi-sets; then, we can define the following relations and operations.*

- *Equality: $\mathcal{A}$ is equal to $\mathcal{B}$ ($\mathcal{A} = \mathcal{B}$) if $A = B$ and $m_A(a) = m_B(a)$, for each $a \in A$.*
- *Inclusion: $\mathcal{A}$ is included in $\mathcal{B}$ ($\mathcal{A} \subset \mathcal{B}$) if $A \subset B$ and $m_A(a) < m_B(a)$, for each $a \in A$. Analogously, $\mathcal{A}$ is included in, or equal to, $\mathcal{B}$ ($\mathcal{A} \subseteq \mathcal{B}$) if $A \subseteq B$ and $m_A(a) \leq m_B(a)$, for each $a \in A$.*
- *Intersection: the intersection of $\mathcal{A}$ and $\mathcal{B}$ ($\mathcal{A} \cap \mathcal{B}$) is a multi-set $\mathcal{C} = (C, m_C)$, such that $C = A \cap B$ and $m_C(c) = \min\{m_A(c), m_B(c)\}$, for each $c \in C$.*
- *Union: the union of $\mathcal{A}$ and $\mathcal{B}$ ($\mathcal{A} \cup \mathcal{B}$) is a multi-set $\mathcal{C} = (C, m_C)$, such that $C = A \cup B$ and $m_C(c) = \max\{m_A(c), m_B(c)\}$, for each $c \in C$.*
- *Sum: the sum of $\mathcal{A}$ and $\mathcal{B}$ ($\mathcal{A} + \mathcal{B}$) is a multi-set $\mathcal{C} = (C, m_C)$, such that $C = A \cup B$ and $m_C(c) = m_A(c) + m_B(c)$, for each $c \in C$.*
- *Difference: the difference of $\mathcal{A}$ and $\mathcal{B}$ ($\mathcal{A} - \mathcal{B}$) is a multi-set $\mathcal{C} = (C, m_C)$, such that $\mathcal{A} = \mathcal{C} + \mathcal{B}$.*

References

1. Brandes, U.; Erlebach, T. (Eds.) *Network Analysis—Methodological Foundations*; Theoretical Computer Science and General Issues; Springer: Berlin/Heidelberg, Germay, 2005; Volume 3418. [CrossRef]
2. Gori, M.; Monfardini, G.; Scarselli, F. A new model for learning in graph domains. In Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, Montreal, QC, Canada, 31 July–4 August 2005; Volume 2, pp. 729–734. [CrossRef]
3. Micheli, A. Neural Network for Graphs: A Contextual Constructive Approach. *IEEE Trans. Neural Netw.* **2009**, *20*, 498–511. [CrossRef]
4. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [CrossRef]
5. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 4–24. [CrossRef]
6. Bruna, J.; Zaremba, W.; Szlam, A.; LeCun, Y. Spectral networks and locally connected networks on graphs. In Proceedings of the International Conference on Learning Representations, Banff, AB, Canada, 14–16 April 2014.
7. Defferrard, M.; Bresson, X.; Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2016; Volume 29, pp. 3844–3852.
8. Kipf, T.; Welling, M. Semi-supervised classification with graph convolutional networks. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
9. Hamilton, W.; Ying, Z.; Leskovec, J. *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
10. Monti, F.; Boscaini, D.; Masci, J.; Rodolà, E.; Svoboda, J.; Bronstein, M.M. Geometric deep learning on graphs and manifolds using mixture model CNNs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 5115–5124.
11. Niepert, M.; Ahmed, M.; Kutzkov, K. Learning Convolutional Neural Networks for Graphs. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; Volume 48, pp. 2014–2023.
12. Gao, H.; Wang, Z.; Ji, S. Large-Scale Learnable Graph Convolutional Networks. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018. [CrossRef]
13. Li, Q.; Han, Z.; Wu, X.M. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
14. Kleinberg, J.; Tardos, E. *Algorithm Design*; Addison-Wesley: Boston, MA, USA, 2005.
15. Dimitrov, N.B.; Morton, D.P. Interdiction Models and Applications. In *Handbook of Operations Research for Homeland Security*; Herrmann, J.W., Ed.; Springer: New York, NY, USA, 2013; pp. 73–103. [CrossRef]
16. Berrone, S.; Della Santa, F.; Pieraccini, S.; Vaccarino, F. Machine learning for flux regression in discrete fracture networks. *GEM-Int. J. Geomath.* **2021**, *12*, 9. [CrossRef]
17. Berrone, S.; Della Santa, F. Performance Analysis of Multi-Task Deep Learning Models for Flux Regression in Discrete Fracture Networks. *Geosciences* **2021**, *11*, 131. [CrossRef]
18. Berrone, S.; Della Santa, F.; Mastropietro, A.; Pieraccini, S.; Vaccarino, F. Discrete Fracture Network insights by eXplainable AI. Machine Learning and the Physical Sciences. In Proceedings of the Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS), Neural Information Processing Systems Foundation, Virtual, 6–7 December 2002. Available online: https://ml4physicalsciences.github.io/2020/ (accessed on 1 February 2022).
19. Berrone, S.; Della Santa, F.; Mastropietro, A.; Pieraccini, S.; Vaccarino, F. Layer-wise relevance propagation for backbone identification in discrete fracture networks. *J. Comput. Sci.* **2021**, *55*, 101458. [CrossRef]
20. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
21. Atwood, J.; Towsley, D. Diffusion-Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2016; Volume 29, pp. 1993–2001.
22. Donon, B.; Donnot, B.; Guyon, I.; Marot, A. Graph Neural Solver for Power Systems. In Proceedings of the International Joint Conference on Neural Networks, Budapest, Hungary, 14–19 July 2019; pp. 1–8. [CrossRef]
23. Donon, B.; Clément, R.; Donnot, B.; Marot, A.; Guyon, I.; Schoenauer, M. Neural networks for power flow: Graph neural solver. *Electr. Power Syst. Res.* **2020**, *189*, 106547. [CrossRef]
24. Raissi, M.; Karniadakis, G.E. Hidden physics models: Machine learning of nonlinear partial differential equations. *J. Comput. Phys.* **2018**, *357*, 125–141. [CrossRef]
25. Raissi, M.; Perdikaris, P.; Karniadakis, George, E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
26. Ding, S. The α-maximum flow model with uncertain capacities. *Appl. Math. Model.* **2015**, *39*, 2056–2063. [CrossRef]
27. Malhotra, V.; Kumar, M.; Maheshwari, S. An O($|V|3$) algorithm for finding maximum flows in networks. *Inf. Process. Lett.* **1978**, *7*, 277–278. [CrossRef]
28. Goldberg, A.V.; Tarjan, R.E. A New Approach to the Maximum-Flow Problem. *J. ACM* **1988**, *35*, 921–940. [CrossRef]
29. Cheriyan, J.; Maheshwari, S.N. Analysis of preflow push algorithms for maximum network flow. In *Foundations of Software Technology and Theoretical Computer Science*; Nori, K.V., Kumar, S., Eds.; Springer: Berlin/Heidelberg, Germay, 1988; pp. 30–48.
30. King, V.; Rao, S.; Tarjan, R. A Faster Deterministic Maximum Flow Algorithm. *J. Algorithms* **1994**, *17*, 447–474. [CrossRef]

31. Goldberg, A.V.; Rao, S. Beyond the Flow Decomposition Barrier. *J. ACM* **1998**, *45*, 783–797. [CrossRef]
32. Golumbic, M.C. (Ed.) *Algorithmic Graph Theory and Perfect Graphs*; Academic Press: Cambridge, MA, USA, 1980.
33. Degiorgi, D.G.; Simon, K. A dynamic algorithm for line graph recognition. In *Graph-Theoretic Concepts in Computer Science*; Nagl, M., Ed.; Springer: Berlin/Heidelberg, Germay, 1995; pp. 37–48. [CrossRef]
34. Barabási, A.L.; Albert, R. Emergence of Scaling in Random Networks. *Science* **1999**, *286*, 509–512. [CrossRef]
35. Albert, R.; Barabási, A.L. Topology of Evolving Networks: Local Events and Universality. *Phys. Rev. Lett.* **2000**, *85*, 5234–5237. [CrossRef]
36. Hagberg, A.A.; Schult, D.A.; Swart, P.J. Exploring Network Structure, Dynamics, and Function using NetworkX. In Proceedings of the 7th Python in Science Conference, Pasadena, CA, USA, 19–24 August 2008; Varoquaux, G., Vaught, T., Millman, J., Eds.; pp. 11–15.
37. Freeman, L.C. A Set of Measures of Centrality Based on Betweenness. *Sociometry* **1977**, *40*, 35–41. [CrossRef]
38. Bavelas, A. Communication Patterns in Task-Oriented Groups. *J. Acoust. Soc. Am.* **1950**, *22*, 725–730. [CrossRef]
39. Sabidussi, G. The centrality index of a graph. *Psychometrika* **1966**, *31*, 581–603. [CrossRef]
40. Erdős, P.; Rényi, A. On Random Graphs. *Publ. Math.* **1959**, *6*, 290–297.
41. Gilbert, E.N. Random Graphs. *Ann. Math. Stat.* **1959**, *30*, 1141–1144. [CrossRef]
42. Reilly, W. *Highway Capacity Manual*; Transport Research Board: Washington, DC, USA, 2000.
43. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on International Conference on Machine Learning, ICML'15, Lille, France, 7–9 July 2015; Volume 37, pp. 448–456.
44. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778. [CrossRef]
45. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *J. Mach. Learn. Res.* **2010**, *9*, 249–256.
46. Kingma, D.P.; Ba, J.L. Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015; pp. 1–15.
47. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software. Available online: tensorflow.org (accessed on 1 February 2022).
48. Adler, P. *Fractures and Fracture Networks*; Kluwer Academic: Dordrecht, The Netherlands, 1999.
49. Cammarata, G.; Fidelibus, C.; Cravero, M.; Barla, G. The Hydro-Mechanically Coupled Response of Rock Fractures. *Rock Mech. Rock Eng.* **2007**, *40*, 41–61. [CrossRef]
50. Fidelibus, C.; Cammarata, G.; Cravero, M. Hydraulic characterization of fractured rocks. In *Rock Mechanics: New Research*; Abbie, M., Bedford, J.S., Eds.; Nova Science Publishers Inc.: New York, NY, USA, 2009.
51. Hyman, J.D.; Hagberg, A.; Osthus, D.; Srinivasan, S.; Viswanathan, H.; Srinivasan, G. Identifying Backbones in Three-Dimensional Discrete Fracture Networks: A Bipartite Graph-Based Approach. *Multiscale Model. Simul.* **2018**, *16*, 1948–1968. [CrossRef]
52. Sanchez-Vila, X.; Guadagnini, A.; Carrera, J. Representative hydraulic conductivities in saturated grqundwater flow. *Rev. Geophys.* **2006**, *44*, 1–46. [CrossRef]
53. Svensk Kärnbränslehantering AB. *Data Report for the Safety Assessment, SR-Site*; Technical Report TR-10-52; SKB: Stockholm, Sweden, 2010.
54. Hyman, J.D.; Aldrich, G.; Viswanathan, H.; Makedonska, N.; Karra, S. Fracture size and transmissivity correlations: Implications for transport simulations in sparse three-dimensional discrete fracture networks following a truncated power law distribution of fracture size. *Water Resour. Res.* **2016**, *52*, 6489. [CrossRef]
55. Berrone, S.; Pieraccini, S.; Scialò, S. A PDE-constrained optimization formulation for discrete fracture network flows. *SIAM J. Sci. Comput.* **2013**, *35*, B487–B510. [CrossRef]
56. Berrone, S.; Pieraccini, S.; Scialò, S. On simulations of discrete fracture network flows with an optimization-based extended finite element method. *SIAM J. Sci. Comput.* **2013**, *35*, A908–A935. [CrossRef]
57. Berrone, S.; Pieraccini, S.; Scialò, S. An optimization approach for large scale simulations of discrete fracture network flows. *J. Comput. Phys.* **2014**, *256*, 838–853. [CrossRef]
58. Srinivasan, S.; Karra, S.; Hyman, J.; Viswanathan, H.; Srinivasan, G. Model reduction for fractured porous media: A machine learning approach for identifying main flow pathways. *Comput. Geosci.* **2019**, *23*, 617–629. [CrossRef]
59. Blizard, W.D. Multiset theory. *Notre Dame J. Form. Log.* **1989**, *30*, 36–66. [CrossRef]
60. Hein, J.L. *Discrete Mathematics*; Jones & Bartlett Publishers: Burlington, MA, USA, 2003.