2009

Computational capabilities of graph neural networks

Franco Scarselli
franco@dii.unisi.it

Marco Gori
marco@dii.unisi.it

Ah Chung Tsoi
Hong Kong Baptist University, act@uow.edu.au

Markus Hagenbuchner
markus@uow.edu.au

Gabriele Monfardini

Computational Capabilities of Graph Neural Networks

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, Gabriele Monfardini

Abstract

In this paper, we will consider the universal approximation properties of a recently introduced neural network model called graph neural network (GNN) which can be used to process structured data inputs, e.g. acyclic graph, cyclic graph, directed or un-directed graphs. This class of neural networks implements a function $\tau(\boldsymbol{G}, n) \in I\!R^m$ that maps a graph $\boldsymbol{G}$ and one of its nodes n onto an m-dimensional Euclidean space. We characterize the functions that can be approximated by GNNs, in probability, up to any prescribed degree of precision. This set contains the maps that satisfy a property, called preservation of the unfolding equivalence, and includes most of the practically useful functions on graphs; the only known exception is when the input graph contains particular patterns of symmetries when unfolding equivalence may not be preserved. The result can be considered an extension of the universal approximation property established for the classic feedforward neural networks. Some experimental examples are used to show the computational capabilities of the proposed model.

Keywords

Graph Neural Networks, Approximation Theory, Universal Approximators, Graphical Domains.

I. Introduction

In a large number of practical and engineering applications, the underlying data is often more conveniently represented in terms of graphs. In fact, a graph naturally represents a set of objects (nodes) and their relationships (edges). For example, in an image, it is natural to represent as nodes regions of the image which have similar intensity or colour, and to represent the relationship among these regions by edges. This is often known as a region adjacency graph. As another example, it is convenient to model the individual web pages as nodes of a graph, and the hyperlink connections among the web pages as edges of the graph.

Traditionally to process graph structured input data, one first "squashes" the graph structure into a vector, and then use neural network models which accept vectorial inputs, e.g. multilayer perceptrons, self organizing maps, to process such resulting data [1]. Such "squashing" of the graph structured input may lose most of the topological relationships among the nodes of the graph. An alternative approach is to preserve the topological relationships among the data items in a graph structured input data, and to follow the graph structure in a node by node processing of the input data [2], [3], [4]. This general approach underpins a number of proposed neural network models, e.g. recursive neural networks, self organizing map for structured data. The advantages of this approach include: the topological relationship among the data items are preserved, and taken into account in the data processing steps; less data processing is required for each node. However, at least in the ways in which the recursive neural network models or the self organizing maps for structured data are formulated [2], [3], they can process limited types of graphs, e.g. acyclic and directed graphs. While recursive neural networks or self organizing maps for structured data can be extended to handle more general graph structures, e.g. cyclic graphs or un-directed graphs or to adopt a more sophisticated processing scheme, e.g. taking into account the ancestors as well as descendants of a node in the processing, they tend to become relatively complicated.

Scarselli, Gori, Monfardini are with the University of Siena, Siena, Italy. Email: {franco,marco,monfardini}@dii.unisi.it.
Tsoi is with Hong Kong Baptist University, Kowloon, Hong Kong. Email: act@hkbu.edu.hk
Hagenbuchner is with University of Wollongong, Wollongong, Australia. Email: markus@uow.edu.au

Recently these various approaches have been unified in a novel neural network model called graph neural networks (GNNs) [5]. GNNs can handle acyclic and cyclic graphs, directed and undirected graphs, and graphs with locally neighborhood dependency. A GNN handles such complexity by deploying two functions in the model: a transition function f, which defines the relationship between the nodes of the graph; and an output function g, which specifies an output for each node. By using these functions, a GNN implements a mapping $\varphi(\mathbf{G}, n) \in I\!R^m$, where $\mathbf{G}$ is a graph, n denotes a node in $\mathbf{G}$, and $I\!R^m$ is the m-dimensional Euclidean space. It was shown empirically that GNNs can be used to model graph structured data, and that trained GNNs can generalize to unforeseen data [6].

However, the approximation capabilities of this model have not been investigated yet and it has not been defined which functions on graphs that GNNs are able to realize. In other words, an interesting question arises: given a generic function $\tau(\mathbf{G}, n) \in I\!R^m$, can it be realized or approximated by a function $\varphi(\cdot, \cdot)$ implemented by a GNN model?

In this paper we will seek to answer this question. In particular, we will show that under mild generic conditions, most of the practically useful functions on graphs can be approximated in probability by GNNs up to any prescribed degree of accuracy. Such a result can be considered an extension of the universal approximation property that was proved for feedforward neural networks [7], [8], [9], [10]. It also extends the universal approximation property of recursive neural networks [11], [12], a neural model which is a predecessor of GNNs and can process only acyclic directed graphs.

The structure of this paper is as follows: after the introduction of some notations used in this paper as well as some preliminary definitions, Section II briefly presents the concept of a graph neural network model. A universal approximation theorem is shown in Section III and the proof of the theorem together with its auxiliary lemmas are given in Section V, while Section IV collects some experimental results on a number of examples used to illustrate the demonstrated property. Finally, conclusions are drawn in Section VI.

II. Graph neural networks

GNN model was proposed in [13], [5]. In this section, we briefly introduce the model and the notation needed in the following of the paper. Readers are referred to [13], [5] for further discussions about the GNN model.

A. Notation

A graph $\boldsymbol{G}$ is a pair $(\boldsymbol{N}, \boldsymbol{E})$, where $\boldsymbol{N}$ is a set of nodes and $\boldsymbol{E}$ is a set of edges (or arcs) between nodes in $\boldsymbol{N}$. Graphs are assumed to be undirected, i.e. for each arc (n, u), the equality $(n, u) = (u, n)$ holds. The set $\mathrm{ne}[n]$ collects the neighbors of n, i.e. the nodes connected to n by an arc, while $\mathrm{co}[n]$ denotes the set of arcs having n as a vertex. Nodes and edges may have labels, which are assumed to be real vectors. The labels attached to node n and edge (n, u) are represented by $\boldsymbol{l}_n \in I\!R^{l_N}$ and $\boldsymbol{l}_{(n,u)} \in I\!R^{l_E}$ respectively, and $\boldsymbol{l}$ is the vector obtained by stacking together all the labels of the graph. The notation adopted for the labels follows a more general scheme. If $\boldsymbol{y}$ is vector that contains data from a graph and S is a subset of its nodes (edges), then $\boldsymbol{y}_S$ is the vector obtained by selecting from $\boldsymbol{y}$ only the components related to the nodes (edges) in S. Thus, for example, $\boldsymbol{l}_{\mathrm{ne}[n]}$ is the vector containing the labels of all the neighbors of n.

Graphs may be either positional or non–positional. The latter are those described so far, while positional graphs differ from them since for each node n, there exists an injective function $\nu_n : \mathrm{ne}[n] \rightarrow \{1, \ldots, |\boldsymbol{N}|\}$ which assigns to each neighbor of n a different position. The position of the neighbor may be important in certain practical applications, e.g. object locations [12].

The graphical domain considered in this paper is the set $\mathcal{D}$ of pairs of graph–node, i.e. $\mathcal{D} = \mathcal{G} \times \mathcal{N}$ where $\mathcal{G}$ is a set of graphs and $\mathcal{N}$ is a subset of their nodes. We assume a supervised learning framework with the learning set $\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{i,j})|, \boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i) \in \mathcal{G},\ n_{i,j} \in \boldsymbol{N}_i,\ \boldsymbol{t}_{i,j} \in I\!R^m,\ 1 \leq i \leq p,\ 1 \leq j \leq q_i\}$, where $n_{i,j}$ denotes the j-th node in the graph $\boldsymbol{G}_i$, $\boldsymbol{t}_{i,j}$ is the desired target associated to $n_{i,j}$. Finally, $p \leq |\mathcal{G}|$ and $q_i \leq |\boldsymbol{N}_i|$. Interestingly, all the graphs of the learning set can be combined into a unique disconnected graph and, therefore, one might think of the learning set as the pair $\mathcal{L} = (\boldsymbol{G}, \mathcal{T})$ where $\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{E})$ is a graph and $\mathcal{T}$ a is set of pairs $\{(n_i, \boldsymbol{t}_i)|\ n_i \in N,\ \boldsymbol{t}_i \in I\!R^m, 1 \leq i \leq q\}$.

B. The model

The intuitive idea underlining the proposed approach is that nodes in a graph represent objects or concepts, and edges represent their relationships. Each concept is naturally defined by its features and the related concepts. Thus, we can attach a *state* $\boldsymbol{x}_n \in I\!R^s$ to each node n, that is based on the information contained in the neighborhood of n (see Figure 1). The variable $\boldsymbol{x}_n$ contains a representation of the concept embodied in node n and can be used to produce an *output* $\boldsymbol{o}_n \in I\!R^m$, i.e. a decision about the concept.

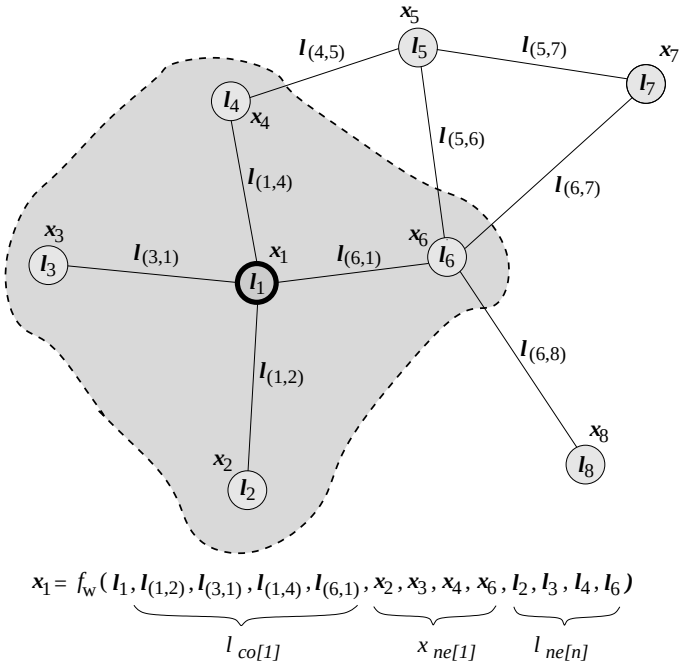


Fig. 1. A graph and the neighborhood of a node. The state $\boldsymbol{x}_1$ of node 1 depends on the information contained in its neighborhood.

Let $f_{\boldsymbol{w}}$ be a parametric function, called *local transition function*, that expresses the dependence of a node n on its neighborhood and let $g_{\boldsymbol{w}}$ be the *local output function* that describes how the output is produced. Then, $\boldsymbol{x}_n$ and $\boldsymbol{o}_n$ are defined as follows

$$\begin{aligned} \boldsymbol{x}_n &= f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}) \\ \boldsymbol{o}_n &= g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)\,, \end{aligned} \tag{1}$$

where $\boldsymbol{l}_n$, $\boldsymbol{l}_{\mathrm{co}[n]}$, $\boldsymbol{x}_{\mathrm{ne}[n]}$, $\boldsymbol{l}_{\mathrm{ne}[n]}$ are respectively the label of n, the labels of its edges, the states and the labels of the nodes in the neighborhood of n. In GNNs, the transition and the output functions are implemented by multilayer feedforward neural network [5].

Remark 1: For the sake of simplicity, only the case of undirected graphs is studied, but the results can be easily extended to directed graphs and even to graphs with mixed directed and undirected arcs. In fact, with minor modifications, GNNs can process

general types of graphs. For example, when dealing with directed graphs, the function f must also accept as an input the direction of each arc, coded, for example, as an additional parameter d_ℓ for each arc $\ell \in \mathrm{co}[n]$ such that $d_\ell = 1$, if ℓ is directed towards n and $d_\ell = 0$, if ℓ comes from n. Moreover, when different kinds of edges co-exist in the same dataset, the label should be designed to distinguish between them. ■

Note that Eq. (1) makes it possible to process both positional and non–positional graphs. For positional graphs, $f_{\boldsymbol{w}}$ needs to receive as additional input the positions of the neighbors. In practice, this can be easily achieved provided that the information contained in $\boldsymbol{x}_{\mathrm{ne}[n]}$, $\boldsymbol{l}_{\mathrm{co}[n]}$, $\boldsymbol{l}_{\mathrm{ne}[n]}$ is sorted according to neighbor positions and is properly padded with special null values in positions corresponding to non–existing neighbors. For example, $\boldsymbol{x}_{\mathrm{ne}[n]} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M]$, where $\boldsymbol{y}_i = \boldsymbol{x}_u$, if u is the i–th neighbor of n ($\nu_n(u) = i$), and $\boldsymbol{y}_i = \boldsymbol{x}_0$, for some predefined null state $\boldsymbol{x}_0$, if there is no i–th neighbor, and $M = \max_{n,u} \nu_n(u)$ is the maximum number of neighbors of the node n.

For non-positional graphs, on the contrary, it is useful to replace function $f_{\boldsymbol{w}}$ of Eq. (1) with

$$\boldsymbol{x}_n = \sum_{u \in \mathrm{ne}[n]} h_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u), \;\; n \in \boldsymbol{N}\,, \tag{2}$$

where $h_{\boldsymbol{w}}$ is a parametric function. In the following, Eq. (2) is referred to as the *Non–positional Form*, while Eq. (1) is called the *Positional Form*. It is worth mentioning that the same structure of Eq. (2) can also be applied to positional graphs provided that the parameters of $h_{\boldsymbol{w}}$ are extended to include a description of the position $\nu_n(u)$ of each neighbor u of n. Formally, positional graphs can be processed when $h_{\boldsymbol{w}}$ takes the position of the neighbors as input, i.e.

$$\boldsymbol{x}_n = \sum_{u \in \mathrm{ne}[n]} h_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u, \nu_n(u)), \;\; n \in \boldsymbol{N}\,, \tag{3}$$

In practical implementations of GNNs and recursive neural networks, the form defined in Eq. (1) is preferred to Eq. (3). However, Eq. (3) is a special case of Eq. (1) and will be particularly useful for proving our results.

Let $\boldsymbol{x}$, $\boldsymbol{o}$, $\boldsymbol{l}$ and $\boldsymbol{l_N}$ be respectively the vectors constructed by stacking all the states, all the outputs, and all the node labels. Then, Eq. (1) can be written in a vectorial form as follows:

$$\begin{aligned} \boldsymbol{x} &= F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}) \\ \boldsymbol{o} &= G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l_N}) \end{aligned} \tag{4}$$

where $F_{\boldsymbol{w}}$ and $G_{\boldsymbol{w}}$ are the composition of $|\boldsymbol{N}|$ instances of $f_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$, respectively. In GNNs, $F_{\boldsymbol{w}}$ is called the *global transition function* while $G_{\boldsymbol{w}}$ the *global output function*. Note that, in order to ensure that $\boldsymbol{x}$ is correctly defined, Eq. (4) must have a unique solution. The Banach Fixed Point theorem [14] provides a sufficient condition for the existence and uniqueness of the solution of such a system of equations. According to Banach's theorem [14], Eq. (4) has a unique solution provided that $F_{\boldsymbol{w}}$ is a *contraction map* with respect to (w.r.t.) the state, i.e. there exists a real number μ, $0 < \mu < 1$, such that $\|F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}) - F_{\boldsymbol{w}}(\boldsymbol{y}, \boldsymbol{l})\| \leq \mu \|\boldsymbol{x} - \boldsymbol{y}\|$ holds for any $\boldsymbol{x}, \boldsymbol{y}$, where $\|\cdot\|$ is any vectorial norm. In GNNs $f_{\boldsymbol{w}}$ is designed so that $F_{\boldsymbol{w}}$ is a contraction map.

Thus, Eq. (1) provides a method to realize a function φ that returns an output $\varphi(\boldsymbol{G}, n) = \boldsymbol{o}_n$ for each graph $\boldsymbol{G}$ and each node n.

Definition 1: HARMOLODIC FUNCTIONS

Let $F_{\boldsymbol{w}}$ be a *contraction map* w.r.t. $\boldsymbol{x}$. Then, any function $\varphi : \mathcal{D} \rightarrow I\!R^m$ generated by $\varphi(\boldsymbol{G}, n) = \boldsymbol{o}_n$ is referred to as a *harmolodic function*[1]. The class of harmolodic functions on $\mathcal{D}$ will be denoted by $\mathcal{H}(\mathcal{D})$. ■

[1]The name "harmolodic function" is inspired by the harmolodic philosophy that is behind jazz music of saxophonist Ornette Coleman.

Banach's Fixed Point theorem suggests also the following classic iterative scheme for computing the value of the stable state

$$\boldsymbol{x}(t+1) = F_{\boldsymbol{w}}(\boldsymbol{x}(t), \boldsymbol{l}) \tag{5}$$

where $\boldsymbol{x}(t)$ denotes the t-th iteration of $\boldsymbol{x}$. This equation converges exponentially fast to the solution of Eq. (4) for any initial value $\boldsymbol{x}(0)$. In fact, Eq. (5) implements the Jacobi iterative method for the solution of non–linear systems [15].

Learning phase in GNN model aims at adapting the parameter set $\boldsymbol{w}$ such that $\varphi_{\boldsymbol{w}}$ approximates the learning set $\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{i,j})|, \boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i) \in \mathcal{G},\ n_{i,j} \in \boldsymbol{N}_i,\ \boldsymbol{t}_{i,j} \in I\!R^m,\ 1 \leq i \leq p,\ 1 \leq j \leq q_i\}$. This learning task can be posed as the minimization of a quadratic error function

$$e_{\boldsymbol{w}} = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (\boldsymbol{t}_{i,j} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))^2 . \tag{6}$$

In GNNs, the minimization is achieved by a new learning algorithm [5] that combines Backpropagation Through Structure algorithm [4], which is used in recursive neural networks, with the Almeida–Pineda algorithm [16], [17]. In order to ensure that the global transition function $F_{\boldsymbol{w}}$ remains a contraction map during learning phase, a penalty term $L\left(\left\|\frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}\right\|\right)$ may be added to the error function (Eq.(6)), where $L(y)$ is $(y-\mu)^2$ if $y > \mu$ and 0 otherwise, and the parameter $\mu \in (0,1)$ defines the desired contraction constant of $F_{\boldsymbol{w}}$.

III. Computational capabilities of GNNs

Feedforward neural networks have been proved to be universal approximators [7], [8], [9] for functions having Euclidean domain and codomain, i.e. they can approximate any map $\tau : I\!R^a \rightarrow I\!R^b$. Several versions of the result have been proposed, which adopt different classes of functions, different measures of the approximation and different network architectures [10]. Recently, also recursive neural networks have been shown to approximate in probability any function on trees up to any degree of precision [11], [12]. More precisely, it has been proved that for any probability measure P, any reals ε, λ and any real function τ defined on trees, there exists a function φ implemented by a recursive neural network such that $P(|\tau(\boldsymbol{T}) - \varphi(\boldsymbol{T})| \geq \varepsilon) \leq 1 - \lambda$ holds. In the following, the approximation capabilities of GNN model are investigated. The analysis presented here concerns the undirected graphs [2] the labels of which are expressed as a vector of reals, i.e. graphs where node labels belong to $I\!R^{l_N}$ and edge labels belong to $I\!R^{l_E}$. Both positional and non–positional GNNs are studied.

In order to discuss the results, some new concepts will be firstly introduced. The purpose is to formally define the class of functions that can be approximated by GNNs. We will demonstrate that this class consists of maps $\tau : \mathcal{D} \rightarrow R^m$ that are generic except for the fact that τ is constrained to produce the same output on nodes that are equivalent according to the relationships specified by the graph, i.e. $n \sim u$ implies $\tau(\boldsymbol{G}, n) = \tau(\boldsymbol{G}, u)$. The equivalence $\sim$ will be called *unfolding equivalence* and will be defined using another concept, the *unfolding tree*, that is formally defined in the following.

An unfolding tree $\boldsymbol{T}_n^d$ is the graph obtained by unfolding $\boldsymbol{G}$ up to the depth d, using n as the starting point (see Figure 2).

Definition 2: UNFOLDING TREE

An unfolding tree $\boldsymbol{T}_n^d$ having depth d of a node n is recursively defined as

$$\boldsymbol{T}_n^d = \begin{cases} \mathrm{Tree}(\boldsymbol{l}_n) & \text{if } d = 1 \\ \mathrm{Tree}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{T}_{\mathrm{ne}[n]}^{d-1}) & \text{if } d > 1 \end{cases}$$

[2]For the sake of simplicity, only the case of undirected graphs is studied. The results can be easily extended to directed graphs.

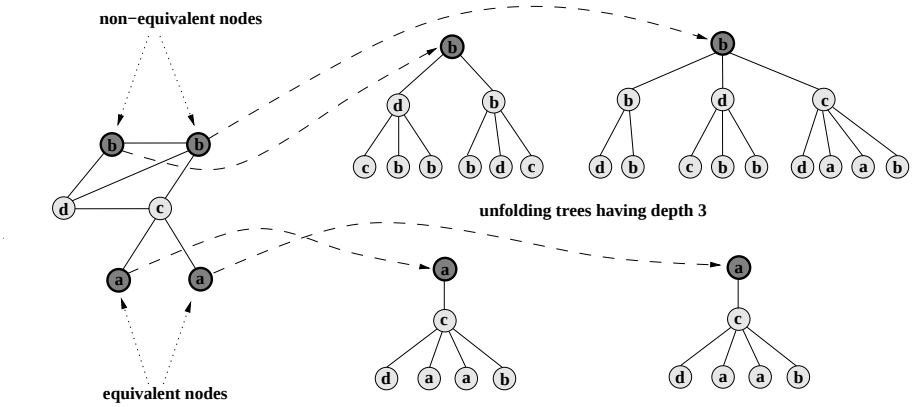


Fig. 2. A graph and four unfolding trees of depth 3. Dashed lines specify the correspondence between a node and its unfolding tree. The two nodes with label b are not unfolding equivalent because their unfolding trees are different, whereas the two nodes with label a are unfolding equivalent.

Here, $\boldsymbol{T}^{d-1}_{\mathrm{ne}[n]} = [\boldsymbol{T}^{d-1}_{u_1}, \boldsymbol{T}^{d-1}_{u_2}, \ldots]$ is the vector containing the unfolding trees having depth $d-1$ of the neighbors $\mathrm{ne}[n] = \{u_1, u_2, \ldots\}$ of n. The operator Tree constructs a tree from the label of the root, the labels of the edges entering into the root and a set of sub–trees [3]. Moreover, the possibly infinite tree $\boldsymbol{T}_n = \lim_{d\to\infty} \boldsymbol{T}^d_n$ that can be constructed by merging all the unfolding trees $\boldsymbol{T}^d_n$ on any d will simply be called the unfolding tree of n. ■

An example of construction of the unfolding tree is shown in Figure 2. Unfolding trees naturally induce an equivalence relationship on the nodes of $\boldsymbol{G}$.

Definition 3: UNFOLDING EQUIVALENCE

Let $\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{E})$ be an undirected graph. The nodes $n, u \in \boldsymbol{N}$ are said to be *unfolding equivalent*, $n \sim u$, if $\boldsymbol{T}_n = \boldsymbol{T}_u$. ■

For example, Figure 2 shows a graph with two unfolding non–equivalent nodes, two unfolding equivalent nodes, and their respective unfolding trees of depth 3. In this particular example, non–equivalent nodes can be immediately distinguished at the first level of the trees, since they have a different number of children.

Functions that do not distinguish nodes which are unfolding equivalent are said *to preserve the unfolding equivalence*.

Definition 4: FUNCTIONS PRESERVING THE UNFOLDING EQUIVALENCE

A function $l : \mathcal{D} \to I\!R^m$ is said to preserve the unfolding equivalence on $\mathcal{D}$, if

$$n \sim u \text{ implies } l(\boldsymbol{G}, n) = l(\boldsymbol{G}, u)$$

The class of functions which preserves the unfolding equivalence on $\mathcal{D}$ is denoted by $\mathcal{F}(\mathcal{D})$. ■

For example, let us apply a given function $l : \mathcal{D} \to I\!R^m$ to the graph in Figure 2. If l preserves the unfolding equivalence, then l is constrained to produce the same output for the two nodes n_1, n_2 having label **a**, i.e. $l(\boldsymbol{G}, n_1) = l(\boldsymbol{G}, n_2)$.

Remark 2: The exact meaning of the given definitions is slightly different according to whether positional or non–positional graphs are to be considered. If the graphs are positional, the unfolding trees should take into account also the original neighbors' positions. Moreover, equation $\boldsymbol{T}_n = \boldsymbol{T}_u$ in Definition 3 uses the equality embedded in positional trees. For non–positional graphs, the unfolding trees and the equality are both non–positional. ■

The following theorem states that functions preserving the unfolding equivalence compute the outputs at a node n considering only the information contained in the unfolding tree $\boldsymbol{T}_n$.

[3] If no sub–tree is given, as in $\mathrm{Tree}(\boldsymbol{l}_n)$, the constructed tree contains only one node.

Theorem 1: FUNCTIONS OF UNFOLDING TREES

A function l belongs to $\mathcal{F}(\mathcal{D})$ if and only if there exists a function κ defined on trees such that $l(\boldsymbol{G}, n) = \kappa(T_n)$ for any node n of the domain $\mathcal{D}$. ■

The proofs of all theorems and corollaries presented in this section have been moved to Section V to improve paper readability. The following corollary, which is an immediate consequence of Theorem 1, suggests that $\mathcal{F}(\mathcal{D})$ is a large class of functions. It can be applied, for example, to all the domains where the labels contain real numbers.

Corollary 1: GRAPHS WITH DISTINCT LABELS

Let $\mathcal{G}$ be the set of the graphs of $\mathcal{D}$ and assume that all the nodes have distinct labels, i.e. $n \neq u$ implies $\boldsymbol{l}_n \neq \boldsymbol{l}_u$ for any nodes n, u of $\mathcal{G}$. Then, any function defined on $\mathcal{D}$ preserves the unfolding equivalence. ■

In the following, we assume that $\mathcal{D}$ is equipped with a probability measure P and an integral operator is defined on the functions from $\mathcal{D}$ onto $\mathbb{R}^m$. In order to clarify how these concepts can be formally defined, note that a graph is specified by its structure and its labels. Since node labels and the possible structures of a graph are enumerable, there exists an enumerable partition $\mathcal{D}_1, \mathcal{D}_2, \ldots$ of the domain $\mathcal{D}$ such that $\mathcal{D}_i = \mathcal{G}_i \times \{n_i\}$ and each set $\mathcal{G}_i$ contains only graphs having the same structure. For each i, a graph $\boldsymbol{G} \in \mathcal{G}_i$ is completely defined by the vector $\boldsymbol{l}_{\boldsymbol{G}}$ formed by stacking all its labels and the set $\boldsymbol{l}_{\mathcal{G}_i} = \{\boldsymbol{l}_{\boldsymbol{G}} | \, \boldsymbol{G} \in \mathcal{G}_i\}$, obtained by collecting all those vectors, is a subset of an Euclidean space, i.e. $\boldsymbol{l}_{\mathcal{G}_i} \subseteq R^{c_i}$. In fact, any measure P on $\mathcal{D}$, when restricted to $\boldsymbol{l}_{\mathcal{G}_i}$, is a measure m_i defined on the linear space R^{c_i}. Thus, P can be formally defined, for each $\overline{\mathcal{D}} \subseteq \mathcal{D}$, as

$$P(\overline{\mathcal{D}}) = \sum_i \alpha_i \cdot P(\overline{\mathcal{D}} \cap \mathcal{D}_i) = \sum_i \alpha_i \cdot m_i(\overline{\mathcal{G}}_i)\,, \tag{7}$$

where $\overline{\mathcal{G}}_i$ is specified by the equality $\overline{\mathcal{G}}_i \times \{n_i\} = \overline{\mathcal{D}} \cap \mathcal{D}_i$ and the α_i is a set of positive numbers such that $\sum_i \alpha_i = 1$ [4]. Moreover, we will define the integral of a function l on $\overline{\mathcal{D}}$ as $\int_{\overline{\mathcal{D}}} l(\boldsymbol{G}, n)\, d\boldsymbol{G}\, dn = \sum_i \int_{\boldsymbol{l}_{\overline{\mathcal{G}}_i}} l(\boldsymbol{G}, n)\, d\boldsymbol{l}_{\boldsymbol{G}}$, where each $\int_{\boldsymbol{l}_{\overline{\mathcal{G}}_i}} l(\boldsymbol{G}, n)\, d\boldsymbol{l}_{\boldsymbol{G}}$ is computed using Lebesgue measure theory [18].

The set $\mathcal{F}(\mathcal{D})$ plays an important role in our analysis. In fact, it will be proved that any measurable function $\tau \in \mathcal{F}(\mathcal{D})$ can be approximated by a GNN in probability. Moreover, the converse holds: all the functions implemented by a GNN preserve the unfolding equivalence [5]. First, the result is proved for positional GNNs.

Theorem 2: APPROXIMATION BY POSITIONAL GNNS

Let $\mathcal{D}$ be a domain that contains positional graphs. For any measurable function $\tau \in \mathcal{F}(\mathcal{D})$ preserving the unfolding equivalence, any norm $\|\cdot\|$ on $\mathbb{R}^m$, any probability measure P on $\mathcal{D}$, and any reals $\varepsilon, \mu, \lambda$, where $\varepsilon > 0$, $0 < \lambda < 1$, $0 < \mu < 1$, there exist two continuously differentiable functions f and g such that

$$\begin{aligned} \boldsymbol{x}_n &= f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}) \\ \boldsymbol{o}_n &= g(\boldsymbol{x}_n, \boldsymbol{l}_n),\ n \in \boldsymbol{N}\,, \end{aligned}$$

the global transition function F is a contraction map with a contracting constant μ, the state dimension is $s = 1$, the stable state is uniformly bounded, and the corresponding harmolodic function defined by $\varphi(\boldsymbol{G}, n) \doteq \boldsymbol{o}_n$ satisfies the condition

$$P(\|\tau(\boldsymbol{G}, n) - \varphi(\boldsymbol{G}, n)\| \geq \varepsilon) \leq 1 - \lambda\,.$$

[4] It is worth mentioning that also the converse holds: in fact, any measure on $\mathcal{D}$ can be represented as in Eq. (7) where $\alpha_i = P(\mathcal{D}_i)$.

[5] This is stated in Theorem 4.

■

Commonly used feedforward neural networks (FNNs) are universal approximators [7], [8], [9], [10] and, obviously, they can also approximate the functions f and g of Theorem 2. However, to perfectly simulate the GNN dynamics, we must consider a restricted class of network architectures that can approximate any function and its derivatives at the same time.

Definition 5: FEEDFORWARD NEURAL NETWORKS SUITABLE TO IMPLEMENT GNNS

A class $\mathcal{Q}$ of feedforward neural networks is said to be suitable to implement GNNs, if for any positive integers a, b, any continuously differentiable function $l \in I\!R^a \rightarrow I\!R^b$ with a bounded support and any real numbers $\varepsilon_1 > 0, \varepsilon_2 > 0$, there exist a function $k_{\boldsymbol{w}}$, implemented by a network in $\mathcal{Q}$ and a set of parameters $\boldsymbol{w}$, such that $\|l(\boldsymbol{x}) - k_{\boldsymbol{w}}(\boldsymbol{x})\| \leq \varepsilon_1$ and $\|\frac{\partial l(\boldsymbol{x})}{\partial \boldsymbol{x}} - \frac{\partial k_{\boldsymbol{w}}(\boldsymbol{x})}{\partial \boldsymbol{x}}\| \leq \varepsilon_2$ hold for any $\boldsymbol{x} \in I\!R^b$. ■

In [19], it is proved that the class of three layered neural networks with activation function σ in the hidden neurons and a linear activation function in the output neurons can approximate any function l and its derivatives on R^a, provided that there exists a linear combination k of scaled shifted rotations of σ such that $\frac{\partial k(\boldsymbol{x})}{\partial \boldsymbol{x}}$ is a square integrable function of uniformly locally bounded variation. It can be easily proved that three layered neural networks using common differentiable activation functions, e.g. $\sigma(x) = 1/(1 + e^{-x})$, $\sigma(x) = (1 - e^{-x})/(1 + e^{x})$ or $\sigma(x) = \mathrm{atan}(x)$, satisfies the above property and are suitable to implement GNNs. The following Corollary proves that f and g can be replaced by networks suitable to implement GNNs without losing the property stated in Theorem 2.

Corollary 2: CONNECTIONIST IMPLEMENTATION OF POSITIONAL GNNS

Let us assume that the hypotheses of Theorem 2 hold, that the nodes of the graph in $\mathcal{D}$ have a bounded number of neighbors and that $\mathcal{Q}$ is a class of networks suitable to implement GNNs. Then, there exists a parameter set $\boldsymbol{w}$ and two functions $f_{\boldsymbol{w}}$ (transition function) and $g_{\boldsymbol{w}}$ (output function) implemented by networks in $\mathcal{Q}$, such that the thesis of Theorem 2 is true. ■

The hypothesis on the boundedness of the number of neighbors is needed because f, without such a constraint, can have any number of inputs, whereas a FNN can only have a predefined number of inputs. It is worth mentioning that the hypothesis could be removed by adopting the form defined in Eq. (3) in place of the one expressed in Eq. (1). In this case we can prove that $h_{\boldsymbol{w}}$ can be implemented by a multilayered FNN [6].

The definition of network class suitable to implement GNNs can be weakened, if we admit that the GNN state $\boldsymbol{x}(t)$ remains bounded during the computation of the fixed point. Such an assumption is reasonable in a real application and can be guaranteed by using a fixed initial state, e.g. $\boldsymbol{x}(0) = 0$. In fact, the proofs of Theorem 2 and Corollary 2 demonstrate that if the states are bounded, g and f have to be approximated only on compact subsets of their domains, instead of the whole domains. With such a simplification, the universal approximation literature provide several other results about the approximation of a function along with its derivatives [20], [21], [10]. For example, in [10], it is proved that three layered networks with non–polynomial analytic activation functions can implement any polynomial on compact sets. Since polynomials are dense in continuous functions also with respect to derivatives, three layered networks with non–polynomial analytic activations are suitable to implement GNNs.

The transition function defined in Eq. (2) is less general than the one in Eq. (1). For this reason, one may wonder whether non–positional GNN based on Eq. (2) has narrower approximation capabilities than the GNN of Eq. (1). The following theorem states that both models have the same computational power.

[6] A formal proof of this statement, which is not included in this paper for space reasons, can be easily obtained by the reasoning of the proof of Corollary 2.

Theorem 3: APPROXIMATION BY NON–POSITIONAL GNNS

Let $\mathcal{D}$ be a domain that contains non–positional graphs. For any measurable function $\tau \in \mathcal{F}(\mathcal{D})$ that preserves the unfolding equivalence, any norm $\|\cdot\|$ on $\mathbb{R}^m$, any probability measure P on $\mathcal{D}$, and any reals $\varepsilon, \mu, \lambda$, where $\varepsilon > 0$, $0 < \lambda < 1$, $0 < \mu < 1$, there exist two continuously differentiable functions h and g such that

$$\begin{aligned} \boldsymbol{x}_n &= \sum_{u \in \mathrm{ne}[n]} h(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u) \\ \boldsymbol{o}_n &= g(\boldsymbol{x}_n, \boldsymbol{l}_n),\ n \in \boldsymbol{N}\,, \end{aligned}$$

the global transition function F is a contraction map with contraction constant μ, the state dimension is $s = 1$, the stable state is uniformly bounded, and the corresponding harmolodic function defined by $\varphi(\boldsymbol{G}, n) \doteq \boldsymbol{o}_n$ satisfies the condition

$$P(\|\tau(\boldsymbol{G}, n) - \varphi(\boldsymbol{G}, n)\| \geq \varepsilon) \leq 1 - \lambda$$

■

In addition, we have the following corollary.

Corollary 3: CONNECTIONIST IMPLEMENTATION OF NON–POSITIONAL GNNS

Let us assume that the hypothesis of Theorem 2 holds and $\mathcal{Q}$ is a class of network suitable to implement GNNs. Then, there exists a parameter set $\boldsymbol{w}$ and two functions $f_{\boldsymbol{w}}$ (transition function) and $g_{\boldsymbol{w}}$ (output function) implemented by networks in $\mathcal{Q}$, such that Theorem 3 holds. ■

Finally, the following theorem proves that a GNN can implement only functions that preserve the unfolding equivalence. Hence, the functions realizable by the proposed model are exactly those described in Theorems 2 and 3 respectively.

Theorem 4: $\mathcal{H}(\mathcal{D}) \subseteq \mathcal{F}(\mathcal{D})$

Let φ be the function implemented by a GNN. If the GNN is positional, then φ preserves the unfolding equivalence on positional graphs, while if the GNN is non positional, then φ preserves the unfolding equivalence on non–positional graphs. ■

Theorems 2, 3 and 4 can be provided with intuitive explanations. GNNs use a local computational framework, i.e. the processing consists of "small jobs" operated on each single node. There is no global activity and two "small jobs" can communicate only if the corresponding nodes are neighbors. The output $\boldsymbol{o}_n = \varphi(\boldsymbol{G}, n)$ of node n depends only on the information contained in its neighbors, and recursively, in all the connected nodes. In other words, $\varphi(\boldsymbol{G}, n)$ is a function of the unfolding tree $\boldsymbol{T}_n$, which, according to Theorem 1 implies that φ preserves the unfolding equivalence.

What the GNNs cannot do is described by the following two cases. Theorems 2, 3 and 4 ensure that GNNs do not suffer from other limitations except for those mentioned here. If two nodes n_1 and n_2 are "completely symmetric" (recursively equivalent) and cannot be distinguished on the basis of information contained in the connected nodes, then a GNN will produce the same output for those nodes. In the example depicted in Figure 3, every node has the same label $\mathbf{a}$ and graphs $\mathbf{A}$ and $\mathbf{B}$ are regular, i.e. each node has exactly the same number of edges. Thus, all the nodes of graph $\mathbf{A}$ (graph $\mathbf{B}$) are "symmetric" and will have the same output, i.e. $\varphi(\boldsymbol{G}, n_1) = \varphi(\boldsymbol{G}, n_2)$ if both n_1 and n_2 belong to $\mathbf{A}$ (or both n_1 and n_2 belong to $\mathbf{B}$). Moreover, GNNs cannot compute general functions on disconnected graphs. If $\boldsymbol{G}$ is composed of disconnected graphs, the information contained in a subgraph cannot influence the output of a node which is not reachable from that subgraph. For example, if n is a node of graph $\mathbf{A}$ in Figure 3, then $\varphi(\boldsymbol{G}, n)$ cannot be influenced by $\mathbf{B}$, e.g. $\varphi(\boldsymbol{G}, n)$ cannot count the number of edges of graph $\mathbf{B}$.

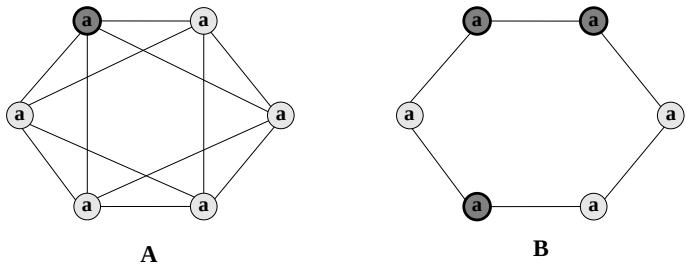


Fig. 3. Two regular graphs where all the nodes have the same label $\mathbf{a}$. Two functions that do not preserve the unfolding equivalence are also displayed. Each function τ is represented by black and white nodes. A node n is black if $\tau(\boldsymbol{G}, n) = 1$ and it is white if $\tau(\boldsymbol{G}, n) = 0$.

It is worth mentioning that in common graph theory all the nodes of a graph are considered different entities. On the contrary, in GNNs two nodes are equal unless the available information suggests otherwise. Such a property is not necessarily a limitation, for two different reasons. Firstly, because it may capture an intuitive idea of the information contained in a graph. In fact, the unfolding tree $\boldsymbol{T}_n$ contains all the data that can be reached by surfing the graph from n. If we assume that the graph defines all available information about the domain objects and their relationships, then it is reasonable to think that $\boldsymbol{T}_n$ describes all our knowledge about n. In addition, the definition of function preserving the unfolding equivalence captures all the reasonable functions on a graphical domain. Secondly, if the considered application requires that some nodes are distinct, then the goal can be practically obtained by inserting into the dataset the appropriate information. Let us consider again the examples depicted in Figure 3. If n is a node of graph $\mathbf{A}$ and $\varphi(\boldsymbol{G}, n)$ should depend on the information contained in $\mathbf{B}$, then there must be some hidden relationship between the object represented by n and the objects represented by the nodes of $\mathbf{B}$. By explicitly representing this relationship with appropriate edges, $\mathbf{A}$ and $\mathbf{B}$ become a connected graph and the GNN model can produce the desired function. Similarly, if some nodes are unfolding equivalent, but φ should produce different outputs, then there exists some information which distinguishes among the equivalent nodes and is not represented in the graph. Including such information into the labels (or in general into the graph) will solve the problem.

The presented theory also extends all the currently known results on approximation capabilities of recursive neural networks (RNNs). In fact, it has been proved that RNNs can approximate in probability any function on trees [11], [12]. On the other hand, when processing a tree, a recursive neural network acts as an GNN where the neighborhood of a node only contains its children, i.e. the father is not included (see [5] for a more detailed comparison). It can be easily observed that under this definition of neighborhood, any function on trees, which satisfies the unfolding equivalence and Theorems 2, 3, reproduces those presented in [11], [12].

Moreover, the concept of unfolding tree has been introduced in [22], where it is used to implement a procedure that allows to process cyclic graphs by RNNs. Such an approach extracts, from the input graph, the unfolding trees of all the nodes: then, those trees are processed by a RNN. It is proved that such a method allows to approximate in probability any function on cyclic graphs with distinct labels. Such a result can now be deduced by using Corollary 1, that functions on graphs with distinct labels preserve the unfolding equivalence and, by Theorem 1, the output of a node can be computed, without loss of generality, using the corresponding unfolding tree.

The intuition delivered by these results is that a wide class of maps on graphs is implementable by a diffusion mechanism based on a transition function and an output function. Here, we also proved that the global transition function can be restricted to be a

contraction map. Such result is crucial for the applications of the GNN model to practical problems using generic forms of graphs, e.g. those generated by multilayered feedforward neural networks, knowing that as long as the underlying function is generic in nature (as the functions which cannot be approximated by the proposed GNNs are pedagogical in nature) can be approximated to a prescribed degree of accuracy. These universal approximation results thus recommend the GNNs as suitable practical models for processing of most classes of graph structured input data, e.g. cyclic, or acyclic, directed or undirected.

IV. Experimental results

This Section presents four experiments designed to demonstrate peculiarities of the GNN model that can be observed in its practical applications and are related to its approximation properties. In the first example it is shown that by adding noise to the node labels of a dataset, we can transform a function that does not preserve the unfolding equivalence to a function that preserves the unfolding equivalence. The experiment demonstrates that such a function, which in theory is approximable by a GNN, can be, even if it is only partially, learned. The other three experiments face problems with different levels of difficulties. Here, the difficulty depends on the complexity of the coding that must be stored in the states. Whereas in theory a GNN can realize most of the functions on graphs, in practice the learnability may be limited by the architecture adopted for the transition function f and the output function g, and by the presence of local minima in the error function. We will observe that the accuracy of the learned function decreases while the coding becomes more complex. Other experiments, whose goal is to assess the performance and the properties of the GNN model on wider and real–life applications, can be found in [5], [23], [24], [6], [25], [26], [27],

The following facts hold for each experiment, unless otherwise specified. The functions involved in the GNN model $g_{\boldsymbol{w}}$, $h_{\boldsymbol{w}}$ were implemented by three layered (one hidden layer) FNNs with sigmoidal activation functions. The presented results were averaged on five different runs. In each run, the dataset was a collection of random graphs constructed by the following procedure: each pair of nodes was connected with a certain probability δ; the resulting graph was checked to verify whether it was connected or not and, finally, if it was not, random edges were inserted until the condition was satisfied. The dataset was split into a training set, a validation set and a test set and the validation set is used to avoid possible issues with overfitting. In every trial, the training procedure performed at most $5,000$ epochs and for every 20 epochs the GNN was evaluated on the validation set. The GNN that achieved the lowest error on validation set was considered the best model and was applied on test set.

The performance of the model is measured by the accuracy in classification problems (when $t_{i,j}$ can take only the values -1 or 1) and by the relative error in regression problems (when $t_{i,j}$ may be any real number). More precisely, in classification problems, a pattern is considered correctly classified if $\varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}) > 0$ and $t_{i,j} = 1$ or if $\varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}) < 0$ and $t_{i,j} = -1$. Thus, accuracy is defined as the percentage of patterns correctly classified by the GNN on the test set. On the other hand, in regression problems, the relative error on a pattern is given by $|(t_{i,j} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))/t_{i,j}|$.

A. Half hot on uniform graphs

This problem consists of learning by examples a relation τ that, given a graph $\boldsymbol{G}_i$, returns $\tau(\boldsymbol{G}_i, n_{i,j}) = 1$ for a half of the nodes of $\boldsymbol{G}_i$ and $\tau(\boldsymbol{G}_i, n_{i,j}) = -1$, for the other half. Figure 3**B** shows an example of τ.

The dataset contained connected regular graphs, i.e. graphs where each node has the same number of connections. As discussed in Section III, if all the labels of the nodes are equal and the graphs are regular, then τ does not preserve the unfolding equivalence and cannot be realized by a GNN. In practice, when a GNN is applied on a regular graph, it produces the same output on each

node. However, the labels can be made distinct by extending them with a random component. With this extension, according to Corollary 1, τ can be realized by a GNN.

The purpose of this experiment is to check the above theoretical results and to verify whether the extension of the labels with random vectors can actually increase the computational power of GNNs. In this experiment, 300 random graphs were equally subdivided into training set, validation set and test set. Each graph $\boldsymbol{G}_i$ was generated by a three step procedure:

Step 1 An even random number of nodes d in the range $[4-10]$ and a random integer number of links c in the range $[3-20]$ were generated. The numbers are produced by uniform probability distributions.

Step 2 A random undirected regular graph with d nodes and c connections for each node was generated. The graph was produced by recursively inserting random edges between nodes which had not reached the maximal number of connections. The construction procedure may be stopped either because a regular graph is obtained or because a configuration was reached where no more edges could be inserted. The construction procedure was repeated until a regular graph was generated.

Step 3 A random node label $\boldsymbol{l}_n$ was attached to each node n. Each label is a 5 dimensional vector containing integers in the range $[0, 5]$.

Note that given a graph $\boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i)$, there are many different functions τ solving the task. However, for our purposes, no particular one is preferable. Such a concept can be expressed applying the following error function

$$e_{\boldsymbol{w}} = \sum_{j,k:\ j \neq k} \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}) \cdot \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,k})\,.$$

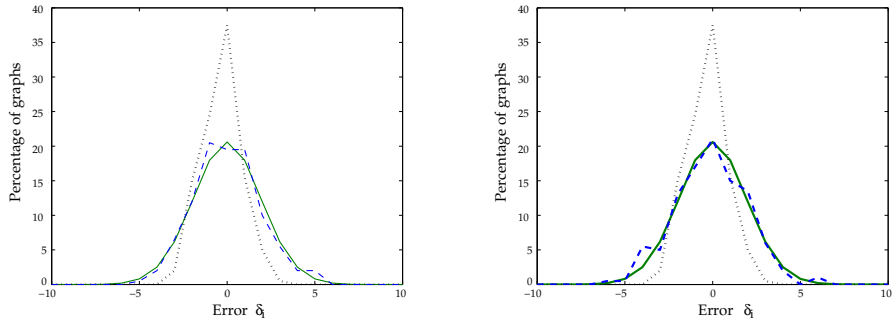
to each graph $\boldsymbol{G}_i$. It can be easily proved that if $\boldsymbol{G}_i$ contains an even number of nodes and $\varphi_{\boldsymbol{w}}$ produces values in the range $[-1, 1]$, then $e_{\boldsymbol{w}}$ reaches a local minima when for half of the nodes $\varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}) = -1$ and $\varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}) = 1$ for the other half.

For this experiment, a GNN was employed where both the transition function $h_{\boldsymbol{w}}$ and the output function $g_{\boldsymbol{w}}$ were implemented by three layered FNNs with 5 hidden neurons. The constraint $o_n \in [-1, 1]$ was enforced using a hyperbolic tangent activation in the output layer of the FNN that implements $g_{\boldsymbol{w}}$.

For each graph $\boldsymbol{G}_i$ of the dataset, the test procedure computed the difference between the desired result and the achieved one as $\delta_i = \frac{|\boldsymbol{G}_i|}{2} - b_i$, where b_i was the number of "hot" nodes. A node n was considered hot if $o_n > 0$. The GNN predicted the correct result, i.e. $\delta_i = 0$, in 38% of the cases. Moreover, for only 2% of the total number of patterns the differences were larger than 2 ($|\delta_i| > 2$). The dotted lines in Figure 4 show the results achieved for each possible value of δ_i on the test set and the training set respectively.

One may argue that the results achieved by GNNs cannot be correctly evaluated without a statistical analysis of the dataset. In fact, even a simple procedure that assigns to each output a random value may often produce the right result, because the case $\delta_i = 0$ is the most probable one. On the other hand, the expected behavior of such a procedure can be easily computed [7] and is depicted in Figure 4 (continuous line). Interestingly, the GNN used the random labels to distinguish nodes and outperforms the random process. Moreover, the results have been compared also with a three layer FNN (dashed line in Figure 4). The FNN was fed only by node labels and did not use graph connectivity. The results obtained by such a network were very similar to those expected for the random procedure. In fact, the experiments have shown the FNN just learns to produce a balanced number of hot and non–hot

[7] Note that the most useful random procedure is the process which sets o_i to a value in $\{-1, 1\}$ with uniform probability. In this case, the probability of producing δ_i hot nodes in a graph with d nodes is $\begin{pmatrix} d \\ \delta_i \end{pmatrix} / 2^d$.



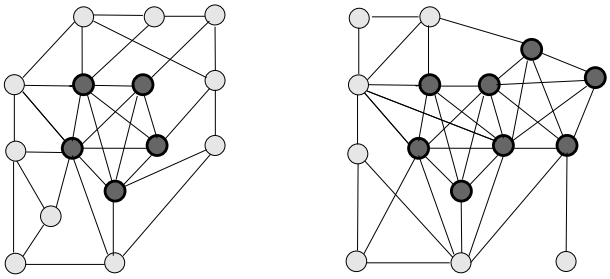
A **B**

Fig. 4. The results achieved on the test set (**A**) and the training set (**B**) for the half hot problem. Horizontal axes display the possible differences δ_i, vertical axes denote the percentage of graphs where the GNN obtained the error δ_i. The dotted, the continuous and the dashed lines represent the results achieved by the GNN, a random process and a FNN, respectively.

nodes in the whole dataset.

B. The clique problem

A *clique* of size k is a complete subgraph with k nodes [8] in a larger graph (see Figure 5). The goal of this experiment was to detect



$\boldsymbol{G}_1$ $\boldsymbol{G}_2$

Fig. 5. Two graphs $\boldsymbol{G}_1, \boldsymbol{G}_2$ that contain one clique and two cliques of five nodes, respectively. Dark gray nodes belong to at least one clique.

cliques of size 5 in the input graphs. More precisely, the GNN was trained to approximate the function defined by $\tau(\boldsymbol{G}_i, n_{i,j}) = 1$, if $n_{i,j}$ belongs to a clique of size 5, and $\tau(\boldsymbol{G}_i, n_{i,j}) = -1$, otherwise. The dataset contained $2,000$ random graphs of 20 nodes each: 300 graphs in the training set, 300 in the validation set, and the rest in the test set. After the construction procedure described at the beginning of this section, a clique of size 5 was inserted into each graph of the dataset. Thus, each graph had at least one clique, but it could have more cliques, due to the random dataset construction. The graph density used in the construction ($\delta = 0.3$) was heuristically selected so as to build a small but not negligible number of graphs with two or more cliques. In fact, only about 65% of the graphs had only five nodes belonging to a clique (the graph contains just one clique), while in some particular cases more than half the nodes of a graph were involved in a clique (Figure 6).

The overall percentage of nodes belonging to a clique was 28.2%. All the nodes were supervised and the desired outputs $t_{i,j} = \tau(\boldsymbol{G}_i, n_{i,j})$ were generated by a brute force algorithm that localized all the cliques of the graphs.

Table I shows the accuracies achieved on this problem by a set of GNNs obtained by varying the number of hidden neurons of

[8] A graph is complete if there is an edge between each pair of nodes.

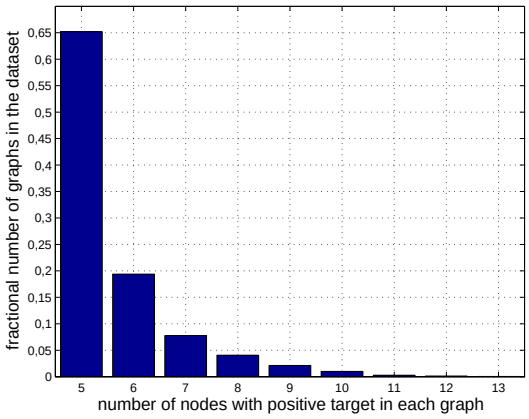


Fig. 6. The distribution of the number of nodes with positive target in the graphs of the dataset.

the FNNs that compose the GNN, i.e. $g_{\boldsymbol{w}}$, $h_{\boldsymbol{w}}$. For the sake of simplicity, the same number of hidden neurons was used in all the FNNs. Finally, the dimension of the state was set to $s = 2$. Some experiments with larger states have shown only a marginal improvement of the performance.

TABLE I

THE RESULTS ON THE CLIQUE PROBLEM. THE TABLE DISPLAYS THE PERFORMANCE ACHIEVED, ON TEST AND TRAINING SETS, WITH DIFFERENT NUMBERS OF HIDDEN NODES IN THE FNNS THAT COMPOSE THE GNN. THE PERFORMANCE IS MEASURED AS THE PERCENTAGE OF NODES THAT HAVE BEEN CORRECTLY CLASSIFIED

Hiddens	**test**	**train**
2	83.73%	83.45%
5	86.95%	86.60%
10	90.74%	90.33%
20	90.20%	89.72%
30	90.32%	89.82%

The accuracy achieved on the test set is very close to the accuracy on training set, with any number of hidden units. This proves that the GNN model did not suffer from overfitting problems on this experiment and that the accuracy is satisfactory even with a reduced number of hidden neurons.

Finally, one may wonder whether the clique problem can be solved by a simpler approach, for example, by a FNN that takes in as input only the number of neighbors $\left|\mathrm{ne}[n_{i,j}]\right|$ of each node $n_{i,j}$. The number of neighbors is informative on the nature of the data; this can be statistically closely correlated with the target $t_{i,j}$. For instance, it is obvious that if $\left|\mathrm{ne}[n_{i,j}]\right| < 5$ then $n_{i,j}$ cannot belong to any clique of size five. Thus, a FNN with one input, 20 hidden [9] and one output neuron was trained to predict $t_{i,j}$ from $\left|\mathrm{ne}[n_{i,j}]\right|$. The accuracy reached by FNN averaged on five runs was 81.56%. As a consequence, GNNs always outperform FNNs, suggesting that GNNs are able to exploit more information from the graph topology than just the number of neighbors.

[9] Increasing the number of hidden neurons did not improved the result significantly.

However, the difference between the performances of the two models, GNNs and FNNs, was not large. The clique task is a difficult problem for GNNs. In fact, in GNN model the computation is localized on the nodes of the graph (see Eq. (1)), while the detection of a clique requires the simultaneous knowledge of the properties of all the nodes involved in the clique. Learning procedure should adapt the parameters so that the transition function $h_{\boldsymbol{w}}$ accumulates the needed information into node states, while the output function $g_{\boldsymbol{w}}$ decodes the states and produces the right answer. Thus, as suggested by the proofs of Theorems 2 and 3, those functions may be very complex and the learning may be difficult.

C. The Neighbors problem

This simple task consists of computing the number of neighbors $|\mathrm{ne}[n]||$ of each node n. Since the information required to compute the desired output is directly available by counting the arcs entering to each node, GNNs are expected to perform much better on this problem than on the clique problem. On the other hand, the peculiarity of this experiment lies in the fact that the dataset consisted of only one single large graph $\boldsymbol{G}$.

In each run of this experiment, one random graph $\boldsymbol{G}$ with 500 nodes was built. The dataset $\mathcal{L}$ contained a pattern $(\boldsymbol{G}, n_j, t_j)$, $t_j = |\mathrm{ne}[n_j]|$, for each node n_j of the graph. The dataset was randomly split into a training set (125 patterns), a validation set (125 patterns), and a test set (250 patterns). The performance was measured by the percentage of the patterns where GNNs achieved a relative error e_r lower than 0.05 and 0.1 respectively. Table II shows that GNNs solve this problem. As the number of hidden neurons in the FNNs becomes larger, so the percentage of the patterns whose prediction is very close to the desired output t_j. For a large number of hidden neurons, most of the patterns are correctly predicted.

TABLE II

RESULTS ON THE NEIGHBORS PROBLEM. THE TABLE DISPLAYS THE PERFORMANCE ACHIEVED WITH DIFFERENT NUMBERS OF HIDDEN NODES IN THE FNNS THAT COMPOSE THE GNN. THE PERFORMANCE IS MEASURED AS THE PERCENTAGE OF THE NODES HAVING RELATIVE ERROR e_r SMALLER THAN TWO THRESHOLDS: 0.05 AND 0.1

Hiddens	**Test**		**Train**	
	$e_r < 0.05$	$e_r < 0.1$	$e_r < 0.05$	$e_r < 0.1$
2	73.64%	77.40%	87.16%	89.36%
5	89.56%	89.76%	99.76%	99.88%
10	90.64%	91.44%	95.40%	95.92%
20	99.04%	99.72%	99.68%	99.92%

D. The second order Neighbors problem

For this experiment, the graph was constructed as in the neighbor problem. Here, the goal is to compute, for each node n, the number of distinct neighbors' neighbors. In other words, the GNN should predict the number of nodes $|\mathrm{nne}[n_j]|$ that are reachable from n_j by a path containing two edges; the nodes that are connected to n_j by several paths must be counted only once and n_j

itself should not be counted [10]. For this reason, this problem is more difficult to learn than the neighbor problem. Table III shows the obtained results. As in the neighbor problem, the error decreases for larger numbers of hidden units. However, in this case, the GNNs can solve the problem only partially and the percentage of patterns with small relative error $e_r < 0.1$ never exceeds 89%.

TABLE III

THE RESULTS ON NEIGHBORS' NEIGHBORS PROBLEM. THE TABLE DISPLAYS THE PERFORMANCE ACHIEVED ON TEST AND TRAINING SETS, WITH DIFFERENT NUMBERS OF HIDDEN NODES IN THE FNNS THAT COMPOSE THE GNN. THE PERFORMANCE IS MEASURED AS THE PERCENTAGE OF THE NODES HAVING RELATIVE ERROR e_r SMALLER THAN TWO THRESHOLDS: 0.05 AND 0.1

Hiddens	**Test**		**Train**	
	$e_r < 0.05$	$e_r < 0.1$	$e_r < 0.05$	$e_r < 0.1$
2	65.96%	81.88%	82.60%	90.64%
5	66.00%	81.64%	82.88%	90.76%
10	66.04%	80.00%	82.88%	90.12%
20	72.48%	88.48%	87.40%	94.12%
30	70.40%	81.08%	88.96%	95.72%

E. The Tree Depth problem

The goal of the task was to compute the depth $d(n_j)$ of each node n_j in a tree, i.e. the length of the path from the root of the tree to node n_j. In each run, the dataset contained one large tree $\boldsymbol{T}$, with $10,000$ nodes. The tree was built starting from the root and attaching to each node a number of children randomly chosen between 0 and 5. Then, the procedure was applied recursively to each leaf until $\boldsymbol{T}$ contained the given number of nodes. If the final tree had less than $10,000$ nodes (this could have happened as nodes may have no children), the construction was repeated. The depth of the trees, measured after the completion of the construction process, usually belonged to the interval $[10, 25]$.

Thus, each dataset consisted of $10,000$ patterns $(\boldsymbol{T}, n_j, t_j)$, where $t_j = d(n_j)/D$ and D is the maximum depth of the tree, i.e. $D = \max_j d(n_j)$. Training set and validation set collected $2,000$ random patterns from the dataset; the remaining $6,000$ patterns constituted the test set.

Intuitively, this task appears to be more difficult than the neighbors problem, but less difficult than neighbor' neighbors problem. In fact, the depth cannot be computed using only the local information as in neighbors problem. On the other hand, the depth of a node depends on the depth of the parent and such a dependence is expressed by a simpler function than in neighbor' neighbors problem. The results achieved in the experiments seem to confirm such an intuitive idea.

[10]More precisely, the desired output t_j was normalized so that it belongs to $[0, 1]$, i.e. $t_j = |\mathrm{nne}[n_j]|/M$, where M is the maximum number of neighbors' neighbors $M = \max_j \mathrm{nne}[n_j]$.

TABLE IV

THE TREE DEPTH PROBLEM. THE TABLE DISPLAYS THE ACCURACIES ACHIEVED ON TEST AND TRAINING SETS, WITH DIFFERENT NUMBERS OF HIDDEN NODES IN THE FNNS THAT COMPOSE THE GNN. THE PERFORMANCE IS MEASURED AS THE PERCENTAGE OF THE NODES HAVING RELATIVE ERROR e_r SMALLER THAN TWO THRESHOLDS: 0.05 AND 0.1

Hiddens	**Test**		**Train**	
	$e_r < 0.05$	$e_r < 0.1$	$e_r < 0.05$	$e_r < 0.1$
2	70.88%	83.93%	90.47%	94.78%
5	77.62%	92.02%	92.67%	97.41%
10	80.24%	93.12%	93.59%	97.86%
20	80.94%	93.89%	93.73%	98.03%
30	81.71%	93.79%	94.02%	98.06%

V. PROOFS

The proofs of the main results can be found in this section.

A. *Proof of Theorem 1:* FUNCTIONS OF UNFOLDING TREES

If there exists κ such that $\varphi(\boldsymbol{G}, n) = \kappa(\boldsymbol{T}_n)$, then $n_1 \sim n_2$ implies

$$l(\boldsymbol{G}, n_1) = \kappa(\boldsymbol{T}_{n_1}) = \kappa(\boldsymbol{T}_{n_2}) = l(\boldsymbol{G}, n_2)\,.$$

On the other hand, if l preserves the unfolding equivalence, then we can define κ as $\kappa(\boldsymbol{T}_n) = l(\boldsymbol{G}, n)$. Note that the above equality is a correct specification for a function. In fact, if $\boldsymbol{T}_{n_1}$ and $\boldsymbol{T}_{n_2}$ are two unfolding trees, then $\boldsymbol{T}_{n_1} = \boldsymbol{T}_{n_2}$ implies $l(\boldsymbol{G}, n_1) = l(\boldsymbol{G}, n_2)$, such that κ is uniquely defined.

B. *Proof of Theorem 2:* APPROXIMATION BY POSITIONAL GNNS

For the sake of simplicity, the theorem will be proved assuming $m = 1$, i.e. $\tau(\boldsymbol{G}, n) \in I\!R$. However, the result is easily extended to the general case when $\tau(\boldsymbol{G}, n) \in I\!R^m$ is a vector. The GNN that satisfies the theorem can be defined by composition of m GNNs, each one approximating a component of $\tau(\boldsymbol{G}, n)$.

According to Theorem 1, there exists a function κ such that $\tau(\boldsymbol{G}, n) = \kappa(\boldsymbol{T}_n)$. Thus, the main idea of the proof consists of designing a GNN which is able to encode the unfolding trees into the node states. The stable state of a node n will be $\boldsymbol{x}_n = \triangledown(\boldsymbol{T}_n)$, where $\triangledown$ is an encoding function that maps trees to real numbers. In this way, the output function g will obtain a representation of $\boldsymbol{T}_n$ by decoding the state and will produce the desired output using κ. Said differently, the recursive activation of f will implement $\triangledown$, and g will implement $\kappa \circ \triangledown^{-1}$, where $\triangledown^{-1}$ is the inverse function of $\triangledown$ and $\circ$ is the function composition operator.

The proof of the theorem is organized into three sections. In the next section, some preliminary lemmas are proved, which allow to restate the theorem in a simpler form. Then, the coding function $\triangledown$ is defined. Finally, it is proved that $\triangledown$ can be implemented by a transition function f and that the corresponding global transition function F is a contraction map.

B.1 Preliminary results

Theorem 2 requires τ be approximated in probability on the whole $\mathcal{D}$, i.e. $P(|\tau(\boldsymbol{G},n)-\varphi(\boldsymbol{G},n)|\leq\varepsilon)\geq 1-\lambda$. The first step of the proof consists of two lemmas which simplify this problem by showing that the theorem can be reduced to a simpler form where the approximation $|\tau(\boldsymbol{G}_i,n_i)-\varphi(\boldsymbol{G}_i,n_i)|\leq\varepsilon$ is achieved just on finite sets of patterns $\{(\boldsymbol{G}_i,n_i)|\,1\leq i\leq v\}$. Moreover, it is also proved that it is sufficient to consider graphs having integer labels only. Formally, Theorem 2 will be reduced to the following theorem.

Theorem 5: For any finite set of patterns $\{(\boldsymbol{G}_i,n_i)|\,\boldsymbol{G}_i\in\mathcal{G},n_i\in\mathcal{N},1\leq i\leq v\}$ where the graphs have integer labels, any function: $\tau:\mathcal{D}\rightarrow I\!R^m$, which preserves the unfolding equivalence, any reals: ε,μ, where $\varepsilon>0$, $0<\mu<1$, there exist two continuously differentiable functions f and g such that for the GNN defined by

$$\begin{aligned}\boldsymbol{x}_n &= f(\boldsymbol{l}_n,\boldsymbol{l}_{\mathrm{co}[n]},\boldsymbol{x}_{\mathrm{ne}[n]},\boldsymbol{l}_{\mathrm{ne}[n]})\\ \boldsymbol{o}_n &= g(\boldsymbol{x}_n,\boldsymbol{l}_n),\ n\in\boldsymbol{N}\,,\end{aligned}$$

the global transition function F is a contraction map with contracting constant μ, the dimension of the state is $s=1$, the stable state is uniformly bounded, and

$$|\tau(\boldsymbol{G}_i,n_i)-\varphi(\boldsymbol{G}_i,n_i)|\leq\varepsilon$$

holds for any i, $1\leq i\leq v$, where φ is the function implemented by the GNN.

The reduction is carried out by proving two lemmas. The first lemma proves that the domain can be divided into small subsets $\mathcal{D}_1,\mathcal{D}_2,..$, such that the graphs in each subset have the same structure and have similar labels (see Figure 7). A finite number of $\mathcal{D}_i$ is sufficient to cover a subset of the domain whose probability is larger $1-\lambda$.

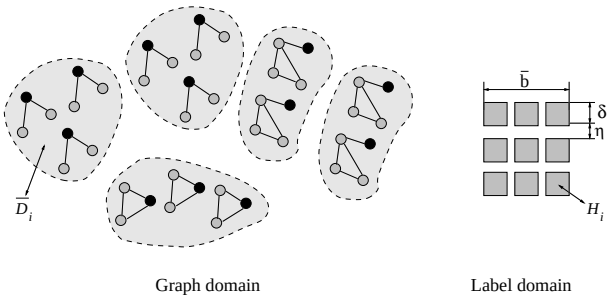


Fig. 7. The partition constructed in Lemma 1. Each subset $\overline{\mathcal{D}}_i$ of the domain contains graphs with the same structure and the same supervised node (the black ones). The labelled domain is divided into hypercubes. The labels of all the graphs in a subset $\overline{\mathcal{D}}_i$ belong to only one hypercube.

Lemma 1: For any probability measure P on $\mathcal{D}$, and any reals λ,δ, where $0<\lambda\leq 1$, $\delta>0$, there exist a real $\bar{b}>0$, which is independent of δ, a set $\overline{\mathcal{D}}\subseteq\mathcal{D}$, and a finite number of partitions $\overline{\mathcal{D}}_1,\ldots,\overline{\mathcal{D}}_v$ of $\overline{\mathcal{D}}$, where $\overline{\mathcal{D}}_i=\mathcal{G}_i\times\{n_i\}$ for graphs $\mathcal{G}_i\subseteq\mathcal{G}$ and a node n_i, such that

1. $P\left(\overline{\mathcal{D}}\right)\geq 1-\lambda$ holds;
2. for each i, all the graphs in $\mathcal{G}_i$ have the same structure, i.e. they differ only in the values of their labels;
3. for each set $\overline{\mathcal{D}}_i$, there exists a hypercube $\mathcal{H}_i\in I\!R^a$ such that $\boldsymbol{l}_{\boldsymbol{G}}\in\mathcal{H}_i$ holds for any graph $\boldsymbol{G}\in\mathcal{G}_i$, where $\boldsymbol{l}_{\boldsymbol{G}}$ denotes the vector obtained by stacking all the labels of $\boldsymbol{G}$;

4. for any two different sets $\mathcal{G}_i, \mathcal{G}_j$, $i \neq j$, their graphs have different structures or their hypercubes $\mathcal{H}_i, \mathcal{H}_j$ have a null intersection $\mathcal{H}_i \cap \mathcal{H}_j = \varnothing$;
5. for each i and each pair of graphs $\boldsymbol{G}_1, \boldsymbol{G}_2 \in \mathcal{G}_i$, the inequality $\|\boldsymbol{l}_{\boldsymbol{G}_1} - \boldsymbol{l}_{\boldsymbol{G}_2}\|_\infty \leq \delta$ holds;
6. for each graph $\boldsymbol{G}$ in $\overline{\mathcal{D}}$, the inequality $\|\boldsymbol{l}_{\boldsymbol{G}}\|_\infty \leq \bar{b}$ holds.

Proof: Two graphs may differ either because of their different structures or because of the different values of their labels. Since the set of the possible structures is enumerable, the set of graphs $\mathcal{G}$ can be partitioned into a sequence of disjoint subsets $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \ldots$, where each $\mathcal{A}_i$ contains graphs having the same structure (they differ only for their label values). Moreover, since there is a finite number of nodes in a graph structure, also $\mathcal{D}$ can be partitioned into a sequence $\mathcal{B}_1, \mathcal{B}_2, \ldots$, where, for each j, $\mathcal{B}_j = \mathcal{C}_j \times \{v_j\}$, $\mathcal{C}_j$ is equal to an $\mathcal{A}_i$ for some i, and v_j is a node of the corresponding graph (structure).

Let η be a real number, $0 < \eta \leq \delta$, b be defined by $b = d\delta$ for some integer d, and $I_i^{b,\eta}$ be the interval $I_i^{b,\eta} = [-b + (i-1)\delta, -b + i\delta - \eta)$, where $1 \leq i \leq 2d$. Moreover, consider all the hypercubes $\mathcal{H}^{b,\eta}$ that can be constructed by taking values in the $I_i^{b,\eta}$, e.g. $\mathcal{H}^{b,\eta} = I_2^{b,\eta} \times I_3^{b,\eta} \times I_2^{b,\eta} \times I_1^{b,\eta}$ is a 4–dimensional hypercube. In the following, we will denote these hypercubes as $\mathcal{H}_1^{b,\eta}, \ldots, \mathcal{H}_j^{b,\eta} \ldots$. Note that each $\mathcal{H}_j^{b,\eta}$ is contained in $[-b, b)^a$, for some a, and their union $\mathcal{H}^{b,\eta} = \bigcup_{j=1}^{2d} \mathcal{H}_j^{b,\eta}$ approximates $\bigcup_{a=1}^{\infty} [b, b)^a$, when $\eta \to 0$. Moreover, for any points $\boldsymbol{l}_1, \boldsymbol{l}_2 \in \mathcal{H}_j^{b,\eta}$, we have $\|\boldsymbol{l}_1 - \boldsymbol{l}_2\|_\infty \leq \delta$, since each interval is shorter than δ.

Let $\mathcal{B}_i^{b,\eta,j}$ be the subset of $\mathcal{B}_i$ containing only the graphs the labels of which belong to $\mathcal{H}_j^{b,\eta}$. Since

$$\lim_{b\to\infty} P\left(\bigcup_{i,j} \mathcal{B}_i^{b,0,j}\right) = P\left(\lim_{b\to\infty}\left(\bigcup_{i,j} \mathcal{B}_i^{b,0,j}\right)\right) = P(\mathcal{D}) = 1\,,$$

there exists $\bar{b}$ such that

$$P\left(\bigcup_{i,j} \mathcal{B}_i^{\bar{b},0,j}\right) \geq 1 - \frac{\lambda}{2}\,. \tag{8}$$

Moreover, since

$$\lim_{k\to\infty,\eta\to 0}\left(\bigcup_{i\leq k,\ j} \mathcal{B}_i^{\bar{b},\eta,j}\right) = \bigcup_{i,j} \mathcal{B}_i^{\bar{b},0,j}\,,$$

there exist $\bar{k}$, $\bar{\eta}$ such that

$$P\left(\bigcup_{i\leq \bar{k},\ j} \mathcal{B}_i^{\bar{b},\bar{\eta},j}\right) \geq P\left(\bigcup_{i,j} \mathcal{B}_i^{\bar{b},0,j}\right) - \frac{\lambda}{2} \geq 1 - \lambda. \tag{9}$$

The sets $\mathcal{B}_i^{\bar{b},\bar{\eta},j}$ involved in Eq. (9) satisfy the properties expected of the sets $\overline{\mathcal{D}}_1, \ldots, \overline{\mathcal{D}}_v$ of the theorem and the $\mathcal{H}_j^{\bar{b},\bar{\eta}}$ are the corresponding hypercubes. In fact, Eq. (9) implies point 1 in the theorem. Points 2, 3, 4 of the theorem follow by definition of the sets $\mathcal{B}_i^{d,\eta,j}$. Moreover, point 5 of the theorem holds because the labels of the graphs in $\mathcal{B}_i^{b,\eta,j}$ belong to the same hypercube $\mathcal{H}_j^{d,\eta}$. Finally, since the labels of the graphs in $\mathcal{B}_i^{\bar{b},\bar{\eta},j}$ are vectors with components in $[-\bar{b}, \bar{b})$, also point 6 of the theorem holds. ■

The following lemma completes the proof of the equivalence between Theorem 2 and Theorem 5. The intuitive idea behind the proof of the theorem is that of constructing a GNN which produces a constant output on each subset $\mathcal{D}_i$. Since there is only a finite number of subsets $\mathcal{D}_i$, Theorem 5 ensures that the construction is possible. Since the $\mathcal{D}_i$ are small and τ is continuous, such a GNN will also satisfy the hypothesis of Theorem 2.

Lemma 2: Theorem 2 holds if and only if Theorem 5 holds.

Proof: Theorem 2 is more general than Theorem 5, so one direction of the implication is straightforward. On the other hand, let us assume that Theorem 5 holds and we have to show that this implies Theorem 2.

Let us apply Lemma 1 setting the values P and λ of the hypothesis equal to the corresponding values of Theorem 2 and δ being any positive real number. It follows that there is a real $\bar{b}$ and a subset $\overline{\mathcal{D}}$ of $\mathcal{D}$ such that $P(\overline{\mathcal{D}}) > 1 - \lambda$. Let $\mathcal{M}$ be the subset of $\mathcal{D}$ that contains only the graphs $\boldsymbol{G}$ satisfying $\|\boldsymbol{l}_{\boldsymbol{G}}\|_\infty \leq \bar{b}$. Note that since $\bar{b}$ is independent of δ, then $\overline{\mathcal{D}} \subset \mathcal{M}$ for any δ.

Since τ is integrable, there exists a continuous function [11] which approximates τ up to any degree of precision in probability. Thus, without loss of generality, we can assume that τ is continuous w.r.t. the labels. Moreover, since $\mathcal{M}$ is bounded, τ is equi-continuous on $\mathcal{M}$. By definition of equi-continuity, a real $\bar{\delta} > 0$ exists such that

$$|\tau(\boldsymbol{G}_1, n) - \tau(\boldsymbol{G}_2, n)| \leq \frac{\varepsilon}{2} \tag{10}$$

holds for any node n and for any pair of graphs $\boldsymbol{G}_1, \boldsymbol{G}_2$ having the same structure and satisfying $\|\boldsymbol{l}_{\boldsymbol{G}_1} - \boldsymbol{l}_{\boldsymbol{G}_2}\|_\infty \leq \bar{\delta}$.

Let us apply Lemma 1 again, where, now, the δ of the hypothesis is set to $\bar{\delta}$, i.e. $\delta = \bar{\delta}$. In the following, $\overline{\mathcal{D}}_i = \mathcal{G}_i \times \{n_i\}$, $1 \leq i \leq v$, represents the sets obtained by the new application of the lemma and $I_i^{\bar{b},\bar{\eta}}$, $1 \leq i \leq 2d$, denotes the corresponding intervals defined in the proof of Lemma 1.

Let $\theta : I\!R \rightarrow I\!N$ be a function that encodes reals into integers as follows: for any i and any $z \in I_i^{\bar{b},\bar{\eta}}$, $\theta(z) = i$. Thus, θ assigns to all the values of an interval $I_i^{\bar{b},\bar{\eta}}$ the index i of the interval itself. Since, the intervals do not overlap (see Figure 7) and are not contiguous, θ can be continuously extended to the entire $I\!R$. Moreover, θ can be extended also to vectors: let $\theta(\boldsymbol{Z})$ denote the vector of integers obtained by coding all the components of $\boldsymbol{Z}$. Finally, let $\Theta : \mathcal{G} \rightarrow \mathcal{G}$ represent the function that transforms each graph by replacing all the labels with their coding, i.e. $\boldsymbol{L}_{\Theta(\boldsymbol{G})} = \theta(\boldsymbol{L}_{\boldsymbol{G}})$.

Let $\overline{\boldsymbol{G}}_1, \ldots, \overline{\boldsymbol{G}}_v$ be graphs, each one extracted from a different set $\mathcal{G}_i$. Note that, according to points 5, 3, 4 of Lemma 1, Θ produces an encoding of the sets $\mathcal{G}_i$. More precisely, for any two graphs $\boldsymbol{G}_1, \boldsymbol{G}_2$ of $\overline{\mathcal{D}}$, we have $\Theta(\boldsymbol{G}_1) = \Theta(\boldsymbol{G}_2)$ if the graphs belong to the same set, i.e. $\boldsymbol{G}_1, \boldsymbol{G}_2 \in \mathcal{G}_i$; and we have $\Theta(\boldsymbol{G}_1) \neq \Theta(\boldsymbol{G}_2)$, otherwise. Thus we can define a function Γ such that $\Gamma(\Theta(\overline{\boldsymbol{G}}_i), n_i) = (\overline{\boldsymbol{G}}_i, n_i)$, $1 \leq i \leq v$.

Consider the problem of approximating $\tau \circ \Gamma$ on the set $(\Theta(\overline{\boldsymbol{G}}_1), n_1), \ldots, (\Theta(\overline{\boldsymbol{G}}_v), n_v)$. Theorem 5 can be applied to such a set, because the set contains a finite number of graphs with integer labels. It follows, that there exists a GNN which implements a function $\bar{\varphi}$ such that, for each i,

$$|\tau(\Gamma(\Theta(\overline{\boldsymbol{G}}_i), n_i)) - \bar{\varphi}(\Theta(\overline{\boldsymbol{G}}_i), n_i)| \leq \frac{\varepsilon}{2}\,. \tag{11}$$

Let f and g be the encoding function and the output function respectively that realise the above GNN. Consider the GNN described by

$$\begin{aligned} \boldsymbol{x}_n &= f(\theta(\boldsymbol{l}_n), \theta(\boldsymbol{l}_{\mathrm{co}[n]}), \boldsymbol{x}_{\mathrm{ne}[n]}, \theta(\boldsymbol{l}_{\mathrm{ne}[n]})) \\ \boldsymbol{o}_n &= g(\boldsymbol{x}_n, \boldsymbol{l}_n)\,. \end{aligned} \tag{12}$$

and let φ be the function implemented by this GNN (Eq. (12)). It is easily shown that for any i and $\boldsymbol{G} \in \mathcal{G}_i$,

$$\varphi(\boldsymbol{G}, n_i) = \bar{\varphi}(\Theta(\overline{\boldsymbol{G}}_i), n_i)$$

holds. Putting together the above equality with Eqs. (10) and (11), it immediately follows, for any $(\boldsymbol{G}, n) \in \overline{\mathcal{D}}_i$

$$\begin{aligned} |\tau(\boldsymbol{G}, n) - \varphi(\boldsymbol{G}, n)| &= |\tau(\boldsymbol{G}, n) - \tau(\overline{\boldsymbol{G}}_i, n) + \\ &\quad \tau(\overline{\boldsymbol{G}}_i, n) - \varphi(\boldsymbol{G}, n)| \end{aligned}$$

[11] Note that the concept of "continuity" is defined only with respect to the labels of the graphs.

$$\begin{aligned} &\leq |\tau(\overline{\boldsymbol{G}}_i, n) - \varphi(\boldsymbol{G}, n)| + \frac{\varepsilon}{2} \\ &= |\tau(\Gamma(\Theta(\overline{\boldsymbol{G}}_i), n)) - \bar{\varphi}(\Theta(\overline{\boldsymbol{G}}_i), n)| \\ &\quad + \frac{\varepsilon}{2} \leq \varepsilon\,. \end{aligned}$$

Thus, the GNN described by Eq. (12) satisfies $|\tau(\boldsymbol{G}, n) - \varphi(\boldsymbol{G}, n)| \leq \varepsilon$ in the restricted domain $\overline{\mathcal{D}}$. Since $P(\overline{\mathcal{D}}) \geq 1 - \lambda$, we have

$$P(|\tau(\boldsymbol{G}, n) - \varphi(\boldsymbol{G}, n)| \leq \varepsilon) \geq 1 - \lambda$$

and the lemma has been shown to be true. ■

B.2 The coding function

The main idea of the proof is that of designing a transition function f which is able to encode the input graph into the node states. In this way, the output function g has only to decode the state and to produce the desired outputs. Of course, the transition function f cannot access directly the whole input graph, but has to read it using the information stored in the states of the neighbor nodes. On the other hand, the target function τ preserves the unfolding equivalence by hypothesis and there exists a function κ such that $\tau(\boldsymbol{G}, n) = \kappa(\boldsymbol{T}_n)$. Thus, an obvious solution will be to store directly the unfolding $\boldsymbol{T}_n$ of node n into the state $\boldsymbol{x}_n$. More precisely, in place of $\boldsymbol{T}_n$, which is infinite and cannot be directly memorized, it is sufficient to store the unfolding up to a depth r, where r is the total number of nodes contained the graphs $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_v$ of Theorem 5. In fact, the following lemma shows that $\boldsymbol{T}_n^r$ is sufficient to define the unfolding equivalence.

Lemma 3: Let us consider the unfolding equivalence $\sim$ defined on a set of graphs $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_v$. For any two nodes u and n, $u \sim n$ holds if and only if $\boldsymbol{T}_u^r = \boldsymbol{T}_n^r$ holds, where $r = \sum_{i=1}^{v} |\boldsymbol{N}_i|$, and $\boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i)$.

Proof: The "only if" part of the proof is straightforward. In fact, by definition, $u \sim n$ implies $\boldsymbol{T}_u^k = \boldsymbol{T}_n^k$, for each k. Thus, $\boldsymbol{T}_u^r = \boldsymbol{T}_n^r$ follows. For the "if part", let us assume $\boldsymbol{T}_u^r = \boldsymbol{T}_n^r$. Note that, for any integer $k \geq 1$, $\boldsymbol{T}_u^k = \boldsymbol{T}_n^k$ implies $\boldsymbol{T}_u^{k-1} = \boldsymbol{T}_n^{k-1}$, because $\boldsymbol{T}_u^{k-1}$ and $\boldsymbol{T}_n^{k-1}$ are subtrees of $\boldsymbol{T}_u^k$ and $\boldsymbol{T}_n^k$, respectively. Thus, there are only three possible cases: (1) $\boldsymbol{T}_u^k = \boldsymbol{T}_n^k$ for any k; (2) $\boldsymbol{T}_u^k \neq \boldsymbol{T}_n^k$ for any k; (3) there exists a $\bar{k} > 1$ such that $\boldsymbol{T}_u^k = \boldsymbol{T}_n^k$, for $k < \bar{k}$ and $\boldsymbol{T}_u^k \neq \boldsymbol{T}_n^k$, for $k \geq \bar{k}$. Case (1) immediately supports our theorem, and case (2) is absurd by the assumption that $\boldsymbol{T}_u^r = \boldsymbol{T}_n^r$ of Theorem 5. Hence, case (2) cannot be true. Let us discuss case (3): we will show that $\bar{k} < r$. If n, u have different (node or edge) labels, their unfolding trees are immediately different at depth 1, i.e. $\boldsymbol{T}_n^1 \neq \boldsymbol{T}_u^1$. On the other hand, if two nodes n, u have the same labels and are connected to the neighbors by edges having the same labels, then $\boldsymbol{T}_n^{\bar{k}} \neq \boldsymbol{T}_u^{\bar{k}}$ may happen only because they have different subtrees which implies that the set of the unfolding trees of the neighbors are different. Putting together the above reasoning with the assumption of case (3), we deduce the following inference rule:

If $\boldsymbol{T}_n^k \neq \boldsymbol{T}_u^k$ and $\boldsymbol{T}_n^{k-1} = \boldsymbol{T}_u^{k-1}$, then there are two neighbors n_2, u_2 of n, u, respectively, for which $\boldsymbol{T}_{n_2}^{k-1} \neq \boldsymbol{T}_{u_2}^{k-1}$ and $\boldsymbol{T}_{n_2}^{k-2} = \boldsymbol{T}_{u_2}^{k-2}$ hold.

Let us consider the equivalence $\sim_k$ defined by $u \sim_k u$ if and only if $\boldsymbol{T}_n^k = \boldsymbol{T}_u^k$, and let us denote by $\equiv$ the equality for equivalences. At the beginning, $\sim_k$ is the largest equivalence, i.e. $u \sim_1 v$ for each u, v having the same label. Then, while k increases, $\sim_k$ becomes more and more refined until $\sim_k$ becomes constant and equals the unfolding equivalence $\sim$. The above inference rule suggests that if $\sim_k \not\equiv \sim_{k-1}$ then $\sim_{k-1} \not\equiv \sim_{k-2}$, i.e. $\sim_{k-1} \equiv \sim_{k-2}$ implies $\sim_k \equiv \sim_{k-1}$. Thus, all the steps k where $\sim_k$ is refined are consecutive. Since, at each refining step at least a class of the equivalence defined by $\sim_k$ is split and the number of

equivalences classes cannot be larger than the number of nodes, then there exist at most $r-1$ refining steps. As a consequence, $\bar{k} < r$ holds. ■

In the following, we describe a representation that will encode trees by real numbers. Such a representation will be used to store the unfolding trees into the states. More precisely, let $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_v$ be the graphs considered in Theorem 5. We will restrict our attention only to the trees up to depth r that can be built from the graphs $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_v$; i.e. the trees $\mathcal{U} = \{\boldsymbol{T}_n^d | \, 1 \leq d \leq r, \, n \, \text{is a node of} \, \boldsymbol{G}_i, 1 \leq i \leq v\}$. Our purpose is that of designing an encoding $\triangledown(\boldsymbol{T}_n^d)$, which maps the tree $\boldsymbol{T}_n^d$ to a real number and is defined for any $\boldsymbol{T}_n^d \in \mathcal{U}$. Function $\triangledown$ will be specified in two steps:

Step 1 A map $\lozenge$ will be defined which assigns a different integer number to each quintuple $(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}_{u_i}^d, \boldsymbol{l}_{u_i})$, where u_i is the i–the neighbor of n. Moreover, the coding $\triangledown$ will be defined as

$$\triangledown(\boldsymbol{T}_n^{d+1}) = \sum_{i=1}^{|\mathrm{ne}[n]|} \gamma^{\lozenge(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}_{u_i}^d, \boldsymbol{l}_{u_i})} \, , \tag{13}$$

where γ is any positive real number smaller than $\frac{Q}{2r(1+Q)}$. Here, Q is given by $Q = \frac{\overline{Q}_1}{\overline{Q}_2}\mu$, where μ is the contraction constant of Theorem 2 (which we are proving), and $\overline{Q}_1, \overline{Q}_2$ are two real numbers such that $\overline{Q}_1 \|\boldsymbol{z}\| \leq \|\boldsymbol{z}\|_1 \leq \overline{Q}_2 \|\boldsymbol{z}\|$ holds for any $\boldsymbol{z}$, the 1–norm $\|\boldsymbol{z}\|_1 = \sum_i |z_i|$, and the norm $\|\cdot\|$ of the hypothesis of Theorem 2 [12].

Step 2 It will be proved that $\triangledown$ is injective on $\mathcal{U}$ and there exists a decoding function $\triangledown^{-1}$ such that $\boldsymbol{T}_n^d = \triangledown^{-1}(\triangledown(\boldsymbol{T}_n^d))$.

The two steps are discussed with more details in the following.

Step 1: function $\lozenge$

Since $\mathcal{U}$ contains a finite number of trees, only a finite number of quintuples $(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}_n^d, \boldsymbol{l}_{u_i})$ exist. So, we can enumerate all the possible quintuples and define the coding $\lozenge$ that assigns a different integer to each quintuple. Among the possible assignments, we select a $\lozenge$ that is monotonically increasing w.r.t. d. More precisely, we assume that for any d and any nodes n, u_i, $\bar{n}$, $\bar{u}_j$,

$$\lozenge(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}_{u_j}^{d+1}, \boldsymbol{l}_{u_i}) > \lozenge(j, \boldsymbol{l}_{\bar{n}}, \boldsymbol{l}_{(\bar{n},\bar{u}_j)}, \boldsymbol{T}_{\bar{u}_j}^d, \boldsymbol{l}_{\bar{u}_j}) \tag{14}$$

holds.

Step 2: the decoding function $\triangledown^{-1}$

Let us consider the function $\xi : \mathcal{U} \rightarrow \mathcal{P}$ that takes in as input an unfolding tree $\boldsymbol{T}_n^d$ and returns the polynomial of the variable γ that is represented on the right side of Eq. (13). Notice that the function ξ is injective on $\mathcal{U}$, because the polynomial $\xi(\boldsymbol{T}_n^d)$ contains a term for each quintuple $(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}_{u_i}^d, \boldsymbol{l}_{u_i})$. In fact, a quintuple contains all the information related to a neighbor of n and is uniquely described by $\lozenge$.

We will show that $\triangledown$ is also injective by using a reduction to absurdity argument. Let us assume that $\triangledown(\boldsymbol{T}_{n_1}^{d_1}) = \triangledown(\boldsymbol{T}_{n_2}^{d_2})$ holds, for some n_1, n_2, d_1, d_2, and that $\boldsymbol{T}_{n_1}^{d_1} = \boldsymbol{T}_{n_2}^{d_2}$ does not hold. By definition, we have $\xi(\boldsymbol{T}_{n_1}^{d_1})(\gamma) = \triangledown(\boldsymbol{T}_{n_1}^{d_1}) = \triangledown(\boldsymbol{T}_{n_2}^{d_2}) = \xi(\boldsymbol{T}_{n_2}^{d_2})(\gamma)$. On the other hand, the polynomial function $\xi(\boldsymbol{T}_{n_1}^{d_1})$ is different from $\xi(\boldsymbol{T}_{n_2}^{d_2})$ because ξ is injective. Thus, γ is a root of the non–null polynomial $\xi(\boldsymbol{T}_{n_1}^{d_1}) - \xi(\boldsymbol{T}_{n_2}^{d_2})$. Such a conclusion cannot be true by the following lemma which shows that if γ is a positive real number, sufficiently close to 0, then γ cannot be a root of $\xi(\boldsymbol{T}_{n_1}^{d_1}) - \xi(\boldsymbol{T}_{n_2}^{d_2})$

Lemma 4: Let $p(x) = \sum_{i=1}^k \alpha_i x^i$ be a polynomial in x with integer coefficients, B be the maximal magnitude of the coefficients, i.e. $B = \max_{i=1}^k |\alpha_i|$. Then p has no root in the open interval $(0, \frac{1}{2B})$.

[12] Such a definition is made possible by the fact that all norms on a finite dimensional space over $\mathbb{R}$ are equivalent.

Proof: Let α_j be the first non–null coefficient, i.e. $\alpha_i = 0, 1 \leq i \leq j-1$, $\alpha_j \neq 0$. Moreover, let us assume $\alpha_j > 0$: the proof when $\alpha_j < 0$ follows by a similar reasoning as shown here. By using simple algebra

$$\begin{aligned} p(x) &= \sum_{i=1}^{k} \alpha_i x^i \\ &\geq \alpha_j x^j - \sum_{i=j+1}^{k} |\alpha_i| x^i \geq x^j - \sum_{i=j+1}^{k} B x^i \\ &= x^j - B x^{j+1} \frac{1-x^{k-j}}{1-x} \\ &= \frac{x^j}{1-x}\left(1 - x - Bx(1-x^{k-j})\right) \\ &= \frac{x^{j+1}(1-x^{k-j})}{1-x}\left(\frac{1-x}{x(1-x^{k-j})} - B\right) \\ &> \frac{x^{j+1}(1-x^{k-j})}{1-x}\left(\frac{2B}{2} - B\right) = 0\,, \end{aligned}$$

where the last inequality follows by the assumption $x \in (0, \frac{1}{2B})$, which implies $1-x > \frac{1}{2}$, and $\frac{1}{x(1-x^{k-j})} > 2B$. Hence the lemma is true. ■

More precisely, note that the coefficients of the polynomial $\xi(\boldsymbol{T}^{d_1}_{n_1}) - \xi(\boldsymbol{T}^{d_2}_{n_2})$ can assume only three numerical values -1, 0, 1. Thus, we can apply Lemma 4 to $\xi(\boldsymbol{T}^{d_1}_{n_1}) - \xi(\boldsymbol{T}^{d_2}_{n_2})$ with $B = 1$. It follows that provided that $0 < \gamma = \frac{Q}{2r(1+Q)} < \frac{1}{2r} \leq \frac{1}{2B}$ holds, ∇ is injective on $\mathcal{U}$ and there exists a decoding function such that $\boldsymbol{T}^d_n = \nabla^{-1}(\nabla(\boldsymbol{T}^d_n))$.

B.3 Implementation of ∇

In this section, we will show how a GNN can implement the coding ∇ and store $\nabla(\boldsymbol{T}^r_n)$ in the state of a node n. In fact, a GNN can construct the coding ∇ recursively storing in the states of larger and larger unfolding trees. At the beginning, the states are set to a predefined initial value which represents a void tree ($\boldsymbol{x}_n(0) = \nabla(\boldsymbol{T}^0)$). Then, the transition function f constructs the representation of a deeper unfolding tree each time the node is activated. In fact, f builds $\nabla(\boldsymbol{T}^{d+1}_n)$, using the set $\nabla(\boldsymbol{T}^d_{\mathrm{ne}[n]}) = [\nabla(\boldsymbol{T}^d_{u_1}), \ldots, \nabla(\boldsymbol{T}^d_{u_{|\mathrm{ne}[n]|}})]$ of the representations stored in the states of the neighbors. The construction process is stopped when the depth r is reached: f is defined so that $\boldsymbol{x}_n(d) = \nabla(\boldsymbol{T}^r_n)$ for each n and $d \geq r$. Thus, our goal is to implement the following transition function

$$f\left(\boldsymbol{l}_n, \nabla(\boldsymbol{T}^d_{\mathrm{ne}[n]}), \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}\right) = \begin{cases} \nabla(\boldsymbol{T}^{d+1}_n) & \text{if} \quad 0 \leq d \leq r-1 \\ \nabla(\boldsymbol{T}^r_n) & \text{if} \quad d \geq r \end{cases}. \tag{15}$$

Such a goal is reached by defining f as

$$f\left(\boldsymbol{l}_n, \boldsymbol{y}, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}\right) = \sum_{i=1}^{|\mathrm{ne}[n]|} h(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, y_i, \boldsymbol{l}_{u_i})\,, \tag{16}$$

where $\boldsymbol{y}$ is the representation of any set of unfolding trees and y_i is the representation of the i-the tree contained in $\boldsymbol{y}$. Moreover, h is the function

$$h(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, z, \boldsymbol{l}_{u_i}) = \begin{cases} \gamma^{\Diamond(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \nabla^{-1}(z), \boldsymbol{l}_u)} & \text{if } 0 \leq d \leq r-1 \\ \gamma^{\Diamond(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \zeta(\nabla^{-1}(z)), \boldsymbol{l}_u)} & \text{if } d \geq r \end{cases}. \tag{17}$$

where γ is the real number in the definition of the coding function ∇ (see Eq. (13)), z is a representation of an unfolding tree and ζ is defined as

$$\zeta(\boldsymbol{T}^r_{u_i}) = \boldsymbol{T}^{r-1}_{u_i}\,,$$

i.e. ζ is a function that extracts from the unfolding tree $\boldsymbol{T}^r_{u_i}$ the tree $\boldsymbol{T}^{r-1}_{u_i}$, which is related to the same node u_i but has a shallower depth [13].

It is easily observed that such a function f satisfies Eq. (15) and realizes the construction of the coding $\nabla(\boldsymbol{T}^d_{u_i})$ as desired. In fact, from Eq. (13),

$$\begin{aligned} f\left(\boldsymbol{l}_n, \nabla(\boldsymbol{T}^d_{u_i}), \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}\right) &= \sum_{i=1}^{|\mathrm{ne}[n]|} h(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \nabla(\boldsymbol{T}^d_{u_i}), \boldsymbol{l}_{u_i}) \\ &= \begin{cases} \sum_{i=1}^{|\mathrm{ne}[n]|} \gamma^{\Diamond(i,\boldsymbol{l}_n,\boldsymbol{l}_{(n,u)},\boldsymbol{T}^d_{u_i},\boldsymbol{l}_u)} = \nabla(\boldsymbol{T}^{d+1}_n) \\ \qquad\qquad \text{if } 0 \le d \le r-1 \\ \sum_{i=1}^{|\mathrm{ne}[n]|} \gamma^{\Diamond(i,\boldsymbol{l}_n,\boldsymbol{l}_{(n,u)},\boldsymbol{T}^{r-1}_{u_i},\boldsymbol{l}_u)} = \nabla(\boldsymbol{T}^r_d) \\ \qquad\qquad \text{if } d \ge r \end{cases} . \end{aligned}$$

On the other hand, f and g are still defined only on a finite set of points, e.g. f is not defined when the first input parameter does not contain a label of a node or the second input parameter is not the coding of a tree. Since we are looking for a differentiable functions, f and g must be extended to accept any vector of reals. Any continuously differentiable extension of g works, because g will operate only on the final stable state. On the other hand, the extension of f must be carefully designed to ensure that the corresponding global transition function F is a contraction map. The following lemma produces the needed results to achieve this goal.

Lemma 5: For any positive real ϑ, there exists a continuously differentiable function $h : \mathbb{R}^{l_\varepsilon+2l_\mathcal{N}+s+1} \to \mathbb{R}$ such that if f is defined as in Eq (16) and F is the global transition function corresponding to f, then

1. Eq. (15) holds for any unfolding tree $\boldsymbol{T}^d_n \in \mathcal{U}$;
2. the inequality

$$\|F(\boldsymbol{x},\boldsymbol{l}) - F(\boldsymbol{y},\boldsymbol{l})\|_1 < r\left(\frac{\gamma}{1-r\gamma} + \vartheta\right)\|\boldsymbol{x}-\boldsymbol{y}\|_1$$

holds for any $\boldsymbol{x}, \boldsymbol{y}$ and any $\boldsymbol{l}$.

Proof: The proof of this lemma is more involved. In order to preserve the flow of the proof of Theorem 2, we will defer the proof until Section V-B.4. ■

In fact, since $0 < \gamma < \frac{Q}{2r(1+Q)}$ by definition of γ, then

$$r\left(\frac{\gamma}{1-r\gamma} + \vartheta\right) = \frac{Q}{2+Q} + \vartheta r < Q$$

holds for a sufficiently small ϑ. As a consequence, the second point of Lemma 5 and the definition of $Q, \overline{Q}_1, \overline{Q}_2$ (see definition of ∇ in page 22) implies

$$\begin{aligned} \|F(\boldsymbol{x},\boldsymbol{l}) - F(\boldsymbol{y},\boldsymbol{l})\| &\le \frac{1}{\overline{Q}_1}\|F(\boldsymbol{x},\boldsymbol{l}) - F(\boldsymbol{y},\boldsymbol{l})\|_1 \\ &< \frac{Q}{\overline{Q}_1}\|\boldsymbol{x}-\boldsymbol{y}\|_1 \\ &\le \frac{Q\overline{Q}_2}{\overline{Q}_1}\|\boldsymbol{x}-\boldsymbol{y}\| \\ &= \mu\|\boldsymbol{x}-\boldsymbol{y}\| . \end{aligned}$$

[13] Note that such a definition is made possible by the fact that an unfolding tree of a given depth d contains the unfolding tree of a shallower depth $d-1$.

Thus, F is a contraction map with contraction constant smaller than μ and Theorem 2 has been proved.

B.4 Proof of Lemma 5

In order to carry out the proof, some properties of the function h and of the coding $\triangledown$ must be considered. The following lemma shows that h behaves as a contraction map with respect to the domain of the trees in $\mathcal{U}$.

Lemma 6: Let h be defined as in (17). For any node n and any integers: $d_1 \geq 1$, $d_2 \geq 1$, $1 \leq i \leq r$, the inequality

$$|h(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \triangledown(\boldsymbol{T}^{d_1}_{u_i}), \boldsymbol{l}_{u_i}) - h(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \triangledown(\boldsymbol{T}^{d_2}_{u_i}), \boldsymbol{l}_{u_i})| \quad \leq \quad \frac{\gamma}{1-r\gamma} |\triangledown(\boldsymbol{T}^{d_1}_{n}) - \triangledown(\boldsymbol{T}^{d_2}_{n})| \,.$$

holds, where u_i is the i–th neighbor of n.

Proof: Without loss of generality, we can assume $r \geq d_2 > d_1 \geq 1$. In fact, the proof of case $r \geq d_1 > d_2 \geq 1$ follows the same reasoning, and, in the case $d_1 = d_2$, it is straightforward. Moreover, by definition of h, the cases $d_1 > r$, $d_2 > r$ can be reduced to $d_1 = r$ and $d_2 = r$, respectively.

In the following, let m_d and M_d be respectively,

$$\begin{aligned} m_d &= \min_{n,u,i} \left(\Diamond(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}^{d}_{u_i}, \boldsymbol{l}_{u_i}) \right) \\ M_d &= \max_{n,u,i} \left(\Diamond(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}^{d}_{u_i}, \boldsymbol{l}_{u_i}) \right), \end{aligned}$$

where $\Diamond$ is the tuple coding function used in h. Since, by definition, $\Diamond$ is monotonically increasing w.r.t. d (see Eq. (14)), then

$$m_{d-1} < M_{d-1} < m_d < M_d$$

and, since $0 < \gamma < 1$,

$$\begin{aligned} \gamma^{M_d} &\leq h(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \triangledown(\boldsymbol{T}^{d}_{u_i}), \boldsymbol{l}_{u_i}) \\ &\leq \gamma^{m_d} < \gamma^{M_{d-1}} \\ &\leq h(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \triangledown(\boldsymbol{T}^{d-1}_{u_i}), \boldsymbol{l}_{u_i}) \\ &\leq \gamma^{m_{d-1}} \end{aligned} \tag{18}$$

holds for any n, i and $d > 1$. Using Eq. (18), it follows

$$\begin{aligned} |\triangledown(\boldsymbol{T}^{d_1}_{n}) - \triangledown(\boldsymbol{T}^{d_2}_{n})| &= \triangledown(\boldsymbol{T}^{d_1}_{n}) - \triangledown(\boldsymbol{T}^{d_2}_{n}) \\ &= \sum_{i=1}^{|\mathrm{ne}[n]|} \gamma^{\Diamond(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}^{d_1-1}_{u_i}, \boldsymbol{l}_{u_i})} - \sum_{i=1}^{|\mathrm{ne}[n]|} \gamma^{\Diamond(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{T}^{d_2-1}_{u_i}, \boldsymbol{l}_{u_i})} \\ &\geq \sum_{i=1}^{|\mathrm{ne}[n]|} \gamma^{M_{d_1-1}} - \sum_{i=1}^{|\mathrm{ne}[n]|} \gamma^{m_{d_2-1}} \\ &\geq \gamma^{M_{d_1-1}} - r\gamma^{m_{d_2-1}} \,. \end{aligned} \tag{19}$$

Moreover, an upper bound on $\gamma^{m_{d_1}}$ is established as

$$\begin{aligned} \gamma^{m_{d_1}} &= \frac{\gamma^{m_{d_1}}}{\gamma^{M_{d_1-1}} - r\gamma^{m_{d_2-1}}} (\gamma^{M_{d_1-1}} - r\gamma^{m_{d_2-1}}) \\ &= \frac{\gamma^{m_{d_1} - M_{d_1-1}}}{1 - r\gamma^{m_{d_2-1} - M_{d_1-1}}} (\gamma^{M_{d_1-1}} - r\gamma^{m_{d_2-1}}) \\ &\leq \frac{\gamma}{1-r\gamma} (\gamma^{M_{d_1-1}} - r\gamma^{m_{d_2-1}}) \,, \end{aligned} \tag{20}$$

where the inequalities $\gamma^{m_{d_1}-M_{d_1-1}} \leq \gamma$ and $\gamma^{m_{d_2-1}-M_{d_1-1}} \leq \gamma$ have been exploited. Finally, that the lemma is true follows by putting together Eqs. (19) and (20)

$$\begin{aligned}
|h(i,\boldsymbol{l}_n,\boldsymbol{l}_{(n,u_i)},\nabla(\boldsymbol{T}_{u_i}^{d_1}),\boldsymbol{l}_{u_i}) - h(i,\boldsymbol{l}_n,\boldsymbol{l}_{(n,u_i)},\nabla(\boldsymbol{T}_{u_i}^{d_2}),\boldsymbol{l}_{u_i})| &= \gamma^{\Diamond(i,\boldsymbol{l}_n,\boldsymbol{l}_{(n,u_i)},\boldsymbol{T}_{u_i}^{d_1},\boldsymbol{l}_{u_i})} - \gamma^{\Diamond(i,\boldsymbol{l}_n,\boldsymbol{l}_{(n,u_i)},\boldsymbol{T}_{u_i}^{d_2},\boldsymbol{l}_{u_i})} \\
&\leq \gamma^{\Diamond(i,\boldsymbol{l}_n,\boldsymbol{l}_{(n,u_i)},\boldsymbol{T}_{u_i}^{d_1},\boldsymbol{l}_{u_i})} \\
&\leq \gamma^{m_{d_1}} \\
&\leq \frac{\gamma}{1-r\gamma}(\gamma^{M_{d_1-1}} - r\gamma^{m_{d_2-1}}) \\
&\leq \frac{\gamma}{1-r\gamma}|\nabla(\boldsymbol{T}_n^{d_1}) - \nabla(\boldsymbol{T}_n^{d_2})|\,.
\end{aligned}$$

■

The next lemma shows that if a function is defined and if it is a contraction map only on a finite set of points, it can be extended to a contraction map on the entire input domain.

Lemma 7: Let η be a positive real number, $l : \mathcal{A} \rightarrow I\!R$ be a function, and $\mathcal{A} \subset I\!R \times I\!R^b$ be a finite set of vectors. Assume that

$$|l(x,\boldsymbol{z}) - l(y,\boldsymbol{z})| \leq \eta|x-y|\,, \tag{21}$$

holds for any vectors $[x,\boldsymbol{z}]$, $[y,\boldsymbol{z}]$ that belong to $\mathcal{A}$, where $x, y \in I\!R$, $\boldsymbol{z} \in I\!R^b$ and $[\cdot,\cdot]$ denotes the operator that stacks two vectors. Then, for any positive real ϑ, l can be extended to the entire $I\!R \times I\!R^b$. The resulting function $\bar{l}$ equals l on $\mathcal{A}$, it is infinitely differentiable and satisfies

$$|\bar{l}(x,\boldsymbol{z}) - \bar{l}(y,\boldsymbol{z})| \leq (\eta+\vartheta)|x-y|\,. \tag{22}$$

on the entire domain, i.e. for any vectors $[x,\boldsymbol{z}]$, $[y,\boldsymbol{z}]$ that belong to $I\!R \times I\!R^b$.

Proof: The proof is carried out in five steps. Each step defines a new function l_i using the previous one: l_{i-1}. The first function is the function l defined by the hypothesis, the last will be the function that satisfies the lemma.

Step 1: EXTENDING $l(x,\boldsymbol{z})$ TO SOME LARGE AND SMALL VALUES x

Let $\mathcal{M} = \{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_v\}$ be the set obtained by removing the first component from each vector in $\mathcal{A}$. For each $\boldsymbol{z}_i$, $\mathcal{A}_i = \{[\boldsymbol{x}_{i,1},\boldsymbol{z}_i],\ldots,[\boldsymbol{x}_{i,k_i},\boldsymbol{z}_i]\}$ denotes the subset of $\mathcal{A}$ that includes all the vectors containing $\boldsymbol{z}_i$. Moreover, for each i, let $x_{i,0}, x_{i,k+1}$ be two real numbers that fulfill

$$\begin{aligned}
x_{i,0} &< \min_{j\neq 0}\left(x_{i,j} - \frac{|l(x_{i,j},\boldsymbol{z}_i)|}{\eta}\right), \\
x_{i,k+1} &> \max_{j\neq 0}\left(\frac{|l(x_{i,j},\boldsymbol{z})|}{\eta} + x_{i,j}\right).
\end{aligned}$$

In the following, $\mathcal{B}_i$ represents the superset of $\mathcal{A}_i$ defined by $\mathcal{B}_i = \mathcal{A}_i \bigcup \{[x_{i,0},\boldsymbol{z}_i],[x_{i,k_i+1},\boldsymbol{z}_i]\}$. The function l_1 is a simple extension of l to $\mathcal{B} = \bigcup_i \mathcal{B}_i$ and is defined by $l_1(x,\boldsymbol{z}) = l(x,\boldsymbol{z})$, if $[x,\boldsymbol{z}] \in \mathcal{A}$, and $l_1(x,\boldsymbol{z}) = 0$, otherwise.

We will prove that l_1 satisfies inequality (21) on $\mathcal{B}$. In fact, this claim holds in a straightforward manner if both $[x,\boldsymbol{z}]$, $[y,\boldsymbol{z}]$ belongs to $\mathcal{A}$. On the other hand, if $[x,\boldsymbol{z}] = [x_{i,0},\boldsymbol{z}_i]$ and $[y,\boldsymbol{z}] = [x_{i,j},\boldsymbol{z}_i]$ for some i, j, then

$$\begin{aligned}
\eta|x_{i,j} - x_{i,0}| &= \eta x_{i,j} - \eta x_{i,0} \\
&= \eta x_{i,j} - \eta \min_{t\neq 0}\left(x_{i,t} - \frac{|l(x_{i,t},\boldsymbol{z}_i)|}{\eta}\right)
\end{aligned}$$

$$\begin{aligned} &\geq |l(x_{i,t}, \boldsymbol{z}_i)| \\ &\geq |l(x_{i,t}, \boldsymbol{z}_i) - l(x_{i,0}, \boldsymbol{z}_i)| . \end{aligned}$$

The proof of the claim follows a similar reasoning for the other cases, i.e. $[x, \boldsymbol{z}] = [x_{i,k+1}, \boldsymbol{z}_i]$, $[y, \boldsymbol{z}] = [x_{i,0}, \boldsymbol{z}_i]$, and $[y, \boldsymbol{z}] = [x_{i,k+1}, \boldsymbol{z}_i]$.

Step 2: EXTENDING $l_1(x, \boldsymbol{z})$ TO ANY x

Without loss of generality, let us assume that, for each i, $x_{i,0}, \ldots, x_{i,k_i+1}$ are sorted according to their values, i.e. $x_{i,j} < x_{i,j+1}$, $0 \leq j \leq k$. Moreover, let $\mathcal{C}$ be defined as $\mathcal{C} = I\!R \times \mathcal{M}$.

The function l_2 generalizes l_1 to the set $\mathcal{C}$. More precisely, l_2 is

$$l_2(x, \boldsymbol{z}_i) = \begin{cases} l_1(x_{i,j}, \boldsymbol{z}_i) + \frac{x - x_{i,j}}{x_{i,j+1} - x_{i,j}} \left(l_1(x_{i,j+1}, \boldsymbol{z}_i) - l_1(x_{i,j}, \boldsymbol{z}_i)\right) \\ \quad \text{if } x_{i,j} \leq x \leq x_{i,j+1},\ 0 \leq j \leq k \\ 0 \\ \quad \text{if } x \leq x_{i,0} \text{ or } x \geq x_{i,k+1} \end{cases}$$

Actually, l_2 is a piecewise linear function on $\mathcal{C}_i = I\!R \times \{\boldsymbol{z}_i\}$ and it equals l_1 on $\mathcal{B}$. Moreover, $|l_2(x, \boldsymbol{z}_i) - l_2(x_{i,j}, \boldsymbol{z}_i)| \leq \eta |x - x_{i,j}|$ holds, because if $j = k + 1$, then $|l_2(x, \boldsymbol{z}_i) - l_2(x_{i,j}, \boldsymbol{z}_i)| = 0$ by definition of l_2, and if $j \leq k$, then

$$\begin{aligned} |l_2(x, \boldsymbol{z}_i) - l_2(x_{i,j}, \boldsymbol{z}_i)| &= |l_1(x_{i,j-1}, \boldsymbol{z}_i) + \frac{x - x_{i,j-1}}{x_{i,j} - x_{i,j-1}} \left(l_1(x_{i,j}, \boldsymbol{z}_i) - l_1(x_{i,j-1}, \boldsymbol{z}_i)\right) - l_1(x_{i,j}, \boldsymbol{z}_i)| \\ &= \left| \frac{x - x_{i,j}}{x_{i,j} - x_{i,j-1}} \left(l_1(x_{i,j}, \boldsymbol{z}_i) - l_1(x_{i,j-1}, \boldsymbol{z}_i)\right) \right| \\ &\leq \eta |x - x_{i,j}| \end{aligned}$$

A similar reasoning can be used to prove $|l_2(x_{i,t}, \boldsymbol{z}_i) - l_2(y, \boldsymbol{z}_i)| \leq \eta |x_{i,t} - y|$.

Let $[x, \boldsymbol{z}_i], [y, \boldsymbol{z}_i]$ be vectors in $\mathcal{C}$, and without loss of generality, assume that $x \geq y$ holds. Let j be the largest index satisfying $x \geq x_{i,j}$, and let t be the smallest index satisfying $y \leq x_{i,t}$. Using Eq. (21) and the inequality $|l_2(x, \boldsymbol{z}_i) - l_2(x_{i,j}, \boldsymbol{z}_i)| \leq \eta |x - x_{i,j}|$, it follows

$$\begin{aligned} |l_2(x, \boldsymbol{z}_i) - l_2(y, \boldsymbol{z}_i)| &\leq |l_2(x, \boldsymbol{z}_i) - l_2(x_{i,j}, \boldsymbol{z}_i)| + |l_2(x_{i,j}, \boldsymbol{z}_i) - l_2(x_{i,t}, \boldsymbol{z}_i)| + |l_2(x_{i,t}, \boldsymbol{z}_i) - l_2(y, \boldsymbol{z}_i)| \\ &\leq \eta |x - x_{i,j}| + |l_1(x_{i,j}, \boldsymbol{z}_i) - l_1(x_{i,t}, \boldsymbol{z}_i)| + \eta |x_{i,t} - y| \\ &\leq \eta |x - x_{i,j}| + \eta |x_{i,j} - x_{i,t}| + |x_{i,t} - y| = \eta |x - y| , \end{aligned}$$

which implies that l_2 satisfies Eq. (21) on $\mathcal{C}$.

Step 3: EXTENDING $l_2(x, \boldsymbol{z})$ TO THE ENTIRE $I\!R \times I\!R^b$

Let $\mathcal{V} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{2^b}\}$ be the vertices of a hypercube $\mathcal{H}$ in $I\!R^b$ that contain the vectors in $\mathcal{M}$ as interior points[14]. By some results shown in [28], $\mathcal{H}$ can be partitioned, by a process called triangulation, into b-simplexes having $\mathcal{M} \cup \mathcal{V}$ as vertices and such that no vector of $\mathcal{M} \cup \mathcal{V}$ is an interior point of a simplex. A b-simplex is a geometric figure having $b+1$ vertices and it is a generalization of a triangle in the domain $I\!R^b$. Each point of a simplex can be obtained as a linear combination of its vertices. Thus, for any $\boldsymbol{z} \in \mathcal{H}$,

[14] A vector will be called an interior point of a polytope if the vector is contained in the polytope, but it is not contained in the polytope's faces.

let us denote by $\mathcal{S}_{\boldsymbol{z}} \subseteq \mathcal{M} \cup \mathcal{V}$ the set of the $b+1$ vertices of the simplex where $\boldsymbol{z}$ is included. Since, a simplex is the convex hull of its vertices, there exist $b+1$ positive reals $\alpha_{\boldsymbol{z},\boldsymbol{v}} \in I\!R$, such that

$$\boldsymbol{z} = \sum_{\boldsymbol{v} \in \mathcal{S}_{\boldsymbol{z}}} \alpha_{\boldsymbol{z},\boldsymbol{v}} \cdot \boldsymbol{v} \qquad \sum_{\boldsymbol{v} \in S_{\boldsymbol{z}}} \alpha_{\boldsymbol{z},\boldsymbol{v}} = 1\,.$$

The function l_3 is defined on the entire $I\!R \times I\!R^b$ as

$$l_3(x,\boldsymbol{z}) = \begin{cases} \sum_{\boldsymbol{v} \in \mathcal{S}_{\boldsymbol{z}}} \alpha_{\boldsymbol{z},\boldsymbol{v}} \cdot l_2(x,\boldsymbol{v}) & \text{if } \boldsymbol{z} \in \mathcal{H} \\ 0 & \text{if } \boldsymbol{z} \notin \mathcal{H} \end{cases}$$

Note that l_3 is a linear function on each simplex and interpolates l_2 on the vertices. Thus, l_3 is piecewise continuous on $\mathcal{H}$. Moreover, l_3 is 0 on the faces of $\mathcal{H}$ and it is 0 outside $\mathcal{H}$. Thus, l_3 is piecewise continuous on $I\!R \times I\!R^b$. Finally, by simple algebra,

$$\begin{aligned} |l_3(x,\boldsymbol{z}) - l_3(y,\boldsymbol{z})| &= \left| \sum_{\boldsymbol{v} \in \mathcal{S}_{\boldsymbol{z}}} \alpha_{\boldsymbol{z},\boldsymbol{v}} \cdot (l_2(x,\boldsymbol{v}) - l_2(y,\boldsymbol{v})) \right| \\ &\leq \sum_{\boldsymbol{v} \in S_{\boldsymbol{z}}} \alpha_{\boldsymbol{z},\boldsymbol{v}} \cdot |(l_2(x,\boldsymbol{v}) - l_2(y,\boldsymbol{v}))| \\ &\leq \sum_{\boldsymbol{v} \in S_{\boldsymbol{z}}} \alpha_{\boldsymbol{z},\boldsymbol{v}} \cdot \eta |x-y| \\ &= \eta |x-y| \end{aligned} \tag{23}$$

which implies that is l_3 satisfies Eq. (21) for any $[x,\boldsymbol{z}] \in I\!R \times I\!R^b$.

Step 4 APPROXIMATING l_3 BY A DIFFERENTIABLE FUNCTION

In the following, ξ will denote an infinitely differentiable probability distribution. We further assume that the support of ξ is inside the unit ball, i.e. $\xi(x,\boldsymbol{z}) = 0$, if $\|[x,\boldsymbol{z}]\| \geq 1$ and ξ is not null in $(0,\mathbf{0})$. Finally, the constants L and P are specified as follows

$$L = \max_{\substack{x,y \in I\!R \\ \boldsymbol{z} \in I\!R^b}} \frac{|\xi(x,\boldsymbol{z}) - \xi(y,\boldsymbol{z})|}{|x-y|} P = 2 \cdot \max_{\substack{[x_{i,j},\boldsymbol{z}_i] \in \mathcal{A} \\ [x_{k,t},\boldsymbol{z}_k] \in \mathcal{A}}} \|[x_{i,j} - x_{k,t}, \boldsymbol{z}_i - \boldsymbol{z}_k]\| \,. \tag{24}$$

Function l_4 will be an infinitely differentiable function that approximates l_3. Let us consider a smoothing operation on l_3 as follows:

$$l_4^{\delta}(x,\boldsymbol{z}) = \int l_3(x-\overline{x}, \boldsymbol{z}-\overline{\boldsymbol{z}}) \sigma_{\delta}(\overline{x},\overline{\boldsymbol{z}}) \overline{x} d\overline{\boldsymbol{z}}\,,$$

where δ is a positive real and the smoothing function σ_{δ} is defined as $\sigma_{\delta}(x,\boldsymbol{z}) = \delta^{b+1} \xi(\frac{x}{\delta}, \frac{\boldsymbol{z}}{\delta})$. According to well known results on convolutions [18], l_4^{δ} is an infinitely differentiable function and $\lim_{\delta \to 0} l_4^{\delta} = l_3$ uniformly. Since the convergence is uniform, there exists $\hat{\delta}$ such that

$$\max_{x,\boldsymbol{z}} |l_4^{\hat{\delta}}(x,\boldsymbol{z}) - l_3(x,\boldsymbol{z})| \leq \frac{\vartheta P \cdot \xi(0,\mathbf{0})}{2L}\,. \tag{25}$$

Thus, we define $l_4 = l_4^{\hat{\delta}}$. Finally, note that by Eq. (23)

$$\begin{aligned} |l_4(x,\boldsymbol{z}) - l_4(y,\boldsymbol{z})| &= \int (l_3(x-\overline{x}, \boldsymbol{z}-\overline{\boldsymbol{z}}) - l_3(y-\overline{x}, \boldsymbol{z}-\overline{\boldsymbol{z}}))\, \sigma_{\bar{\delta}}(\overline{x},\overline{\boldsymbol{z}}) d\overline{x} d\overline{\boldsymbol{z}} \\ &\leq \eta\,|x-y| \cdot \int \sigma(\overline{x},\overline{\boldsymbol{z}}) d\overline{x} d\overline{\boldsymbol{z}} \\ &= \eta\,|x-y|\,, \end{aligned} \tag{26}$$

holds, so that l_4 fulfills Eq. (21) on $\mathbb{R} \times \mathbb{R}^b$.

Step 5:ADJUST THE FUNCTION ON $\mathcal{A}$ FOR AN INTERPOLATION

Note that l_4 is differentiable, but it does not interpolate l on $\mathcal{A}$ anymore. Function l_5 will be an infinitely differentiable map that interpolates l on $\mathcal{A}$. More precisely, l_5 is built by slightly changing l_4 in the neighborhood of the points of $\mathcal{A}$

$$l_5(x, \boldsymbol{z}) = l_4(x, \boldsymbol{z}) + \sum_{[x_{i,j}, \boldsymbol{z}_i] \in \mathcal{A}} \frac{l_3(x_{i,j}, \boldsymbol{z}_i) - l_4(x_{i,j}, \boldsymbol{z}_i)}{\xi(0, \mathbf{0})} \xi\left(\frac{x - x_{i,j}}{P}, \frac{\boldsymbol{z} - \boldsymbol{z}_i}{P}\right)$$

Note that, since ξ is null outside the unit ball and P is twice the maximal distance of the points in $\mathcal{A}$ (see Eq. (24)), then $\xi(\frac{x-x_{i,j}}{P}, \frac{\boldsymbol{z}-\boldsymbol{z}_i}{P}) > 0$ holds only if $[x_{i,j}, \boldsymbol{z}_i]$ is the point of $\mathcal{A}$ closest to $[x, \boldsymbol{z}]$. Thus, for any $x, \boldsymbol{z}$, at most one term of those involved in the sum of Eq. (27) is non–null. Since $[x_{i,j}, \boldsymbol{z}_i] \in \mathcal{A}$ is the closest point to itself, then

$$\begin{aligned} l_5(x_{i,j}, \boldsymbol{z}_i) &= l_4(x_{i,j}, \boldsymbol{z}_i) + \frac{l_3(x_{i,j}, \boldsymbol{z}_i) - l_4(x_{i,j}, \boldsymbol{z}_i)}{\xi(0, \mathbf{0})} \xi(0, \mathbf{0}) \\ &= l_3(x_{i,j}, \boldsymbol{z}_i) = l(x_{i,j}, \boldsymbol{z}_i) \end{aligned}$$

holds.

Finally, let $[x, \boldsymbol{z}]$, and $[y, \boldsymbol{z}]$ be vectors in $\mathbb{R} \times \mathbb{R}^b$. Then, by definition of l_5 and Eqs. (26) and (25),

$$\begin{aligned} &|\cancel{l_5}(x, \boldsymbol{z}) - l_5(y, \boldsymbol{z})| \\ &\quad = |l_4(x, \boldsymbol{z}) - l_4(y, \boldsymbol{z}) + \sum_{[x_{i,j}, \boldsymbol{z}_i] \in \mathcal{A}} \frac{(l_3(x_{i,j}, \boldsymbol{z}_i) - l_4(x_{i,j}, \boldsymbol{z}_i))}{\xi(0, \mathbf{0})} \cdot \left(\xi\left(\frac{x - x_{i,j}}{P}, \frac{\boldsymbol{z} - \boldsymbol{z}_i}{P}\right) - \xi\left(\frac{y - x_{i,j}}{P}, \frac{\boldsymbol{z} - \boldsymbol{z}_i}{P}\right)\right)| \\ &\quad \leq |l_4(x, \boldsymbol{z}) - l_4(y, \boldsymbol{z})| + \sum_{[x_{i,j}, \boldsymbol{z}_i] \in \mathcal{A}} \frac{|l_3(x_{i,j}, \boldsymbol{z}_i) - l_4(x_{i,j}, \boldsymbol{z}_i)|}{\xi(0, \mathbf{0})} \cdot \left|\xi\left(\frac{x - x_{i,j}}{P}, \frac{\boldsymbol{z} - \boldsymbol{z}_i}{P}\right) - \xi\left(\frac{y - x_{i,j}}{P}, \frac{\boldsymbol{z} - \boldsymbol{z}_i}{P}\right)\right| \\ &\quad \leq \eta|x - y| + \frac{\vartheta P}{2L} \cdot \sum_{[x_{i,j}, \boldsymbol{z}_i] \in \mathcal{A}} \left|\xi(\frac{x - x_{i,j}}{P}, \frac{\boldsymbol{z} - \boldsymbol{z}_i}{P}) - \xi(\frac{y - x_{i,j}}{P}, \frac{\boldsymbol{z} - \boldsymbol{z}_i}{P})\right| \end{aligned}$$

Again, since ξ is null outside the unit ball and P is twice the maximal distance of the points in $\mathcal{A}$, there are at most two $[x_{i,j}, \boldsymbol{z}_i] \in \mathcal{A}$ for which $|\xi(\frac{x-x_{i,j}}{P}, \frac{\boldsymbol{z}-\boldsymbol{z}_i}{P}) - \xi(\frac{y-x_{i,j}}{P}, \frac{\boldsymbol{z}-\boldsymbol{z}_i}{P})| \neq 0$ holds. Moreover, the definition of L implies $|\xi(\frac{x-x_{i,j}}{P}, \frac{\boldsymbol{z}-\boldsymbol{z}_i}{P}) - \xi(\frac{y-x_{i,j}}{P}, \frac{\boldsymbol{z}-\boldsymbol{z}_i}{P})| \leq \frac{L}{P}|x - y|$. Thus,

$$|l_5(x, \boldsymbol{z}) - l_5(y, \boldsymbol{z})| \leq \eta|x - y| + \vartheta|x - y| = (\eta + \vartheta)|x - y|$$

and Lemma 7 has been proved with $\bar{l} = l_5$.

■

Proof of Lemma 5

Now, we can proceed with the proof of Lemma 5. To avoid confusion, let us use an alternative notation to represent the function h in (16): l is

$$l(y, \boldsymbol{l}_{n \leftarrow u_i}) = h(i, \boldsymbol{l}_n, \boldsymbol{l}(n, u_i), y, \boldsymbol{l}_{u_i}),$$

where $\boldsymbol{l}_{n \leftarrow u_i}$ collects into a vector the values $i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{l}_{u_i}$ and $y = \nabla(\boldsymbol{T}^d_{u_i})$. Note that according to the specification of h, function l is defined only for the labels and the unfolding tree of a node of the graphs $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_v$ of Section V-B.2.

By Lemma 6,

$$|l(x, \boldsymbol{l}_{n \leftarrow u_i}) - l(y, \boldsymbol{l}_{n \leftarrow u_i})| \leq \frac{\gamma}{1 - r\gamma}|x - y|$$

holds for any $x = \triangledown(\boldsymbol{T}^{d_1}_{u_i})$, $y = \triangledown(\boldsymbol{T}^{d_2}_{u_i})$, $d_1, d_2 \in I\!R$. Moreover, by Lemma 7, l can be extended to an infinitely differentiable function $\bar{l}$ that satisfies

$$|\bar{l}(x, \boldsymbol{l}_{n\leftarrow u_i}) - \bar{l}(y, \boldsymbol{l}_{n\leftarrow u_i})| \leq (\frac{\gamma}{1-r\gamma} + \vartheta)|x - y| \tag{27}$$

for any positive real ϑ, any $x, y \in I\!R$ and any $\boldsymbol{l}_{n\leftarrow u_i} \in I\!R^b$, $b = 2l_N + l_E$. Thus, let f be defined as in Eq. (16), with its parameters being any value in the corresponding Euclidean spaces, i.e.

$$\begin{aligned} f\left(\boldsymbol{l}_n, \boldsymbol{y}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}\right) &= \sum_{i=1}^{|\mathrm{ne}[n]|} \bar{h}(i, \boldsymbol{l}_n, \boldsymbol{l}_{(n,u_i)}, \boldsymbol{y}_{u_i}, \boldsymbol{l}_{u_i}) \\ &= \sum_{i=1}^{|\mathrm{ne}[n]|} \bar{l}(\boldsymbol{y}_{u_i}, \boldsymbol{l}_{n\leftarrow u_i}) \,, \end{aligned}$$

for any $\boldsymbol{y} \in I\!R^a$, $a = s|\boldsymbol{N}|$, and any $\boldsymbol{l} \in I\!R^b$, $b = 2l_N|\boldsymbol{N}| + 2l_E|\boldsymbol{E}|$. Here, $\bar{h}$ is the extension of h represented by $\bar{l}$.

It is clear that function f fulfills point (1) of Lemma 5 by definition of h. On the other hand, by Eq. (27),

$$\begin{aligned} |f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}) - f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{y}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})| &= \left| \sum_{i=1}^{|\mathrm{ne}[n]|} l(\boldsymbol{y}_{u_i}, \boldsymbol{l}_{n\leftarrow u_i}) - l(\boldsymbol{x}_{u_i}, \boldsymbol{l}_{n\leftarrow u_i}) \right| \\ &\leq \sum_{1\leq i\leq r} \left(\frac{\gamma}{1-r\gamma} + \vartheta\right) (|\boldsymbol{x}_{u_i} - \boldsymbol{x}_{u_i}|) \\ &\leq \left(\frac{\gamma}{1-r\gamma} + \vartheta\right) \|\boldsymbol{x}_{\mathrm{ne}[n]} - \boldsymbol{y}_{\mathrm{ne}[n]}\|_1 \,, \end{aligned}$$

holds for any $\boldsymbol{x}, \boldsymbol{y} \in I\!R^a$. Thus, if F is the global transition function corresponding to f, then

$$\begin{aligned} \|F(\boldsymbol{x}, \boldsymbol{l}) - F(\boldsymbol{y}, \boldsymbol{l})\|_1 &= \sum_n |f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}) - f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{y}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})| \\ &\leq r\left(\frac{\gamma}{1-r\gamma} + \vartheta\right) \|\boldsymbol{x} - \boldsymbol{y}\|_1 \,, \end{aligned}$$

holds, and hence point (2) of Lemma 5 has been proved. ■

C. *Proof of Corollary 2:* CONNECTIONIST IMPLEMENTATION OF POSITIONAL GNNS

Let $\varphi_{f,g}$ denote the function realized by a GNN, where f and g are the local transition and the local output functions respectively. Moreover, for any function l, let $\|l\|_\infty$ represent the superior norm, i.e. $\|l\|_\infty = \sup_x \|l(x)\|$. The following lemma proves that $\varphi_{f,g}$ depends continuously on f and g w.r.t. the superior norm.

Lemma 8: Let $\varphi_{f,g}$ be the function realized by a GNN. Suppose that f and g are continuously differentiable, g has a bounded support and the global transition function F is a contraction map. Then, for any real $\zeta > 0$, there exist two reals $\delta_f, \delta_g > 0$ such that

$$\|\varphi_{f,g} - \varphi_{\bar{f},\bar{g}}\|_\infty \leq \zeta$$

holds for any $\varphi_{\bar{f},\bar{g}}$ implemented by a GNN, provided that the corresponding global transition function $\bar{F}$ is a contraction, and the local transition and the local output functions fulfill

$$\|g - \bar{g}\|_\infty \leq \delta_g \text{ and } \|f - \bar{f}\|_\infty \leq \delta_f \,,$$

respectively.

Proof: Since g is continuous and has a bounded support, then it is equi-continuous. Moreover, also G is equi-continuous, because it is built by stacking copies of g. Thus, there exists a real $\Delta > 0$ such that $\|\boldsymbol{x}-\bar{\boldsymbol{x}}\| \leq \Delta$ implies $\|G(\boldsymbol{x},\boldsymbol{l}_N)-G(\bar{\boldsymbol{x}},\boldsymbol{l}_N)\| \leq \zeta/2$, for any $\boldsymbol{x},\bar{\boldsymbol{x}}$.

Let us define $\delta_f = ((1-\eta)\Delta/\|\mathbb{1}_M\|$, where η is the contraction constant of F, $\mathbb{1}_M \in I\!R^M$ is a vector whose components are one, i.e. $\mathbb{1}_M = [1,1,\ldots,1]$, $M = \max_{n\in\mathcal{N}} |\mathrm{ne}[n]|$ is the maximum number of neighbors for a node [15]. Moreover, assume that $\|f-\bar{f}\|_\infty \leq \delta_f$ holds. Note that, since F and $\bar{F}$ consist of stacking copies of f and $\bar{f}$ respectively, then $\|F-\bar{F}\|_\infty \leq \delta_f\|\mathbb{1}_M\|$ holds. Let $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ denote the corresponding fixed points, for a given input graph, of F and $\bar{F}$ respectively. By simple algebra,

$$\begin{aligned}
\|\boldsymbol{x}-\bar{\boldsymbol{x}}\| &= \|F(\boldsymbol{x},\boldsymbol{l})-\bar{F}(\bar{\boldsymbol{x}},\boldsymbol{l})\| \\
&= \|F(\boldsymbol{x},\boldsymbol{l})-F(\bar{\boldsymbol{x}},\boldsymbol{l})+F(\bar{\boldsymbol{x}},\boldsymbol{l})-\bar{F}(\bar{\boldsymbol{x}},\boldsymbol{l})\| \\
&\leq \|F(\boldsymbol{x},\boldsymbol{l})-F(\bar{\boldsymbol{x}},\boldsymbol{l})\| + \|F(\bar{\boldsymbol{x}},\boldsymbol{l})-\bar{F}(\bar{\boldsymbol{x}},\boldsymbol{l})\| \\
&\leq \eta\|\boldsymbol{x}-\bar{\boldsymbol{x}}\| + \delta_f\|\mathbb{1}_M\|\,,
\end{aligned}$$

and, as a consequence,

$$\|\boldsymbol{x}-\bar{\boldsymbol{x}}\| \leq \frac{\delta_f\|\mathbb{1}_M\|}{1-\eta} = \Delta$$

holds. By definition of Δ, it follows

$$\|G(\boldsymbol{x},\boldsymbol{l}_N)-G(\bar{\boldsymbol{x}},\boldsymbol{l}_N)\| \leq \frac{\zeta}{2}\,.$$

Moreover, let us define $\delta_g = \zeta/(2\|\mathbb{1}_M\|)$. Then,

$$\begin{aligned}
\|G(\boldsymbol{x},\boldsymbol{l}_N)-\bar{G}(\bar{\boldsymbol{x}},\boldsymbol{l}_N)\| &= \|G(\boldsymbol{x},\boldsymbol{l}_N)-G(\bar{\boldsymbol{x}},\boldsymbol{l}_N)+G(\bar{\boldsymbol{x}},\boldsymbol{l}_N)-\bar{G}(\bar{\boldsymbol{x}},\boldsymbol{l}_N)\| \\
&\leq \|G(\boldsymbol{x},\boldsymbol{l}_N)-G(\bar{\boldsymbol{x}},\boldsymbol{l}_N)\| + \|G(\bar{\boldsymbol{x}},\boldsymbol{l}_N)-\bar{G}(\bar{\boldsymbol{x}},\boldsymbol{l}_N)\| \\
&\leq \frac{\zeta}{2} + \delta_g\|\mathbb{1}_M\| = \zeta\,.
\end{aligned}$$

which implies $\|\varphi_{f,g}-\varphi_{\bar{f},\bar{g}}\|_\infty \leq \zeta$. ■

Let f and g be the local transition function and local output function of the GNN, as defined in Theorem 2. According to the theorem, $P(\|\tau(\boldsymbol{G},n)-\varphi_{f,g}(\boldsymbol{G},n)\| \geq \varepsilon/2) \leq 1-\lambda$ and $\|\frac{\partial F}{\partial \boldsymbol{x}}\| \leq \eta/2$ hold. Moreover, according to the proof of the theorem, f has a bounded support (see how f is extended to the entire input domain in the proof of Lemma 7). Finally, we can also assume that g has bounded support, because it is an extension of a function defined on a finite set of points (see discussion on page 24).

Let us apply Lemma 8 to $\varphi_{f,g}$ with $\zeta = \varepsilon/2$. By definition of $\mathcal{Q}$, we can assume, without loss of generality, that the functions $\bar{f}$, $\bar{g}$ of the lemma are implemented by networks in $\mathcal{Q}$. Moreover, we can also assume that the Jacobian of $\bar{f}$ approximates the Jacobian of f with precision ϑ. Then, there exist two functions $\bar{f}$ and $\bar{g}$, implemented by FNNs, such that

$$\|\varphi_{f,g}(\boldsymbol{G},n)-\varphi_{\bar{f},\bar{g}}(\boldsymbol{G},n)\| \leq \frac{\varepsilon}{2}\,.$$

for any graph $\boldsymbol{G}$ and node n. As a consequence, it follows:

$$\begin{aligned}
P(\|\tau(\boldsymbol{G},n)-\varphi_{\bar{f},\bar{g}}(\boldsymbol{G},n)\| \geq \varepsilon) &\leq P(\|\tau(\boldsymbol{G},n)-\varphi_{f,g}(\boldsymbol{G},n)\| + \|\varphi_{f,g}(\boldsymbol{G})-\varphi_{\bar{f},\bar{g}}(\boldsymbol{G},n)\| \geq \varepsilon) \\
&\leq P(\|\tau(\boldsymbol{G},n)-\varphi_{f,g}(\boldsymbol{G},n)\| \geq \frac{\varepsilon}{2}) \leq 1-\lambda\,,
\end{aligned}$$

[15] Such a maximum exists according to the hypothesis of Corollary 2.

that is $\varphi_{\bar{f},\bar{g}}$ can approximate τ up to any degree of precision in probability.

Moreover, since, in our setting, all the norms are equivalent there exists a constant Q such that

$$\begin{aligned}\left\|\frac{\partial \bar{F}}{\partial \boldsymbol{x}}(\boldsymbol{G},\boldsymbol{l}_N)\right\| &\leq \left\|\frac{\partial F}{\partial \boldsymbol{x}}\right\| + \left\|\frac{\partial \bar{F}}{\partial \boldsymbol{x}} - \frac{\partial F}{\partial \boldsymbol{x}}\right\| \\ &\leq \frac{\eta}{2} + Q\left\|\frac{\partial \bar{F}}{\partial \boldsymbol{x}} - \frac{\partial F}{\partial \boldsymbol{x}}\right\|_1 \\ &\leq \frac{\eta}{2} + Q\vartheta_2\,.\end{aligned}$$

As a consequence, it is sufficient to set $\vartheta \leq \eta/(2Q)$ in order to ensure that $\bar{F}$ is a contraction map (with contraction constant smaller than η). Thus, the corollary is shown to be true.

D. Proof of Theorem 3 and Corollary 3: APPROXIMATION BY NON–POSITIONAL GNNS AND THE CONNECTIONIST IMPLEMENTATION

The proof of Theorem 3 follows the same reasoning as the proof of Theorem 2 with few minor differences in the definition of the function $\Diamond$ (Section V-B.2, Step 1) and in the demonstration of the existence of a decoding function $\triangledown^{-1}$ (Section V-B.2, Step 2). In fact, in the definition of $\Diamond$, we must take into account that the processed graphs are non–positional. Such a difference can be overcome by discarding the neighbor position from the input parameters of $\Diamond$ [16]. Thus, $\Diamond$ will be defined as a function that is monotonically increasing w.r.t. d and produces a different integer $\Diamond(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{T}_n^d, \boldsymbol{l}_u)$ for each different value of $\boldsymbol{l}_n$, $\boldsymbol{l}_{(n,u)}$, $\boldsymbol{T}_n^d$, and $\boldsymbol{l}_u$.

Moreover, also the proof of the existence of a decoding function $\triangledown^{-1}$ must be changed due to the different definition of $\Diamond$, and, as a consequence, of $\triangledown$. However, an inspection of the proof indicates out that the new definition of $\triangledown$ affects only the maximum coefficient B of the polynomial $\xi(\boldsymbol{T}_{n_1}^{d_1}) - \xi(\boldsymbol{T}_{n_2}^{d_2})$. In fact, B was equal to 1 in Theorem 2, whereas it will be shown that $B \leq r$ in the current case. On the other hand, B affects only Lemma 4, which still holds if $B \leq r$, because for the lemma to be true it is sufficient that $0 < \gamma \leq \frac{1}{2B}$ holds and, in this case, we have $0 < \gamma = \frac{Q}{2r(1+Q)} < \frac{1}{2r} = \frac{1}{2B}$.

Thus, in order to prove Theorem 3, we have only to demonstrate $B \leq r$. Note that each neighbor of n is represented by a term of the polynomial $\xi(\boldsymbol{T}_n^d)$. In this case, it is different from Theorem 2 in that several children may be represented by the same term since the position of the child is not considered. More precisely, this happens when two neighbors u_i, u_j of n have the same unfolding tree, i.e. $\boldsymbol{T}_{u_i}^{d-1} = \boldsymbol{T}_{u_j}^{d-1}$. Intuitively, such an occurrence is not a problem, since the coefficient corresponding to each term of $\xi(\boldsymbol{T}_n^d)$ will count the number of subtrees of a given "type" and such information is sufficient to reconstruct the original non–positional tree $\boldsymbol{T}_n^d$. Formally, since B is the maximum coefficient of the polynomial $\xi(\boldsymbol{T}_{n_1}^{d_1}) - \xi(\boldsymbol{T}_{n_2}^{d_2})$, B cannot be larger than the maximum number of possible trees $\boldsymbol{T}_{u_i}^{d-1}$, which is smaller than the number of neighbors of n. As a consequence, $B \leq |\mathrm{ne}[n]| \leq r$ holds.

Finally, Corollary 3 can be demonstrated using the same argument used in the proof of Corollary 2. In fact, the proof of Corollary 3 shows that a GNN can approximate another GNN, provided that we can approximate up to any degree of precision the transition function f and its derivatives by a network in $\mathcal{Q}$. Similarly, in non–positional GNNs, the function h is approximated by a network in $\mathcal{Q}$. It turns out that, for each n,

$$\|\bar{f}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}) - f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})\| =$$

[16] As a consequence, the neighbor position will be removed from h which has been specified using $\Diamond$.

$$\begin{aligned} &= \| \sum_{u \in \mathrm{ne}[n]} \bar{h}(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u) - \sum_{u \in \mathrm{ne}[n]} h(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u) \| \\ &\leq \sum_{u \in \mathrm{ne}[n]} \| \bar{h}(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u) - h(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u) \| \leq |\mathrm{ne}[n]| \vartheta_1 \end{aligned}$$

holds, where $\bar{h}$ is the function implemented by the neural network, $\bar{f}$ the corresponding transition function, and ϑ_1 is a bound on the achievable accuracy. Since the accuracy is proportional to the number of neighbors, it may appear that f cannot be approximated up to any desired accuracy. On the contrary, we can observe that the function implemented by the GNN does not actually approximate the target function τ on the whole domain $\mathcal{D}$, but only on graphs having a finite set of structures as defined by Theorem 5. Thus, we can concentrate our attention only on those graphs and we can assume that $\mathrm{ne}[n]$ is bounded. As a consequence, f can be approximated up to any degree of precision by implementing h with a network in $\mathcal{Q}$ and a similar reasoning applies also to the approximation of the Jacobian of f.

E. Proof of Theorem 4: $\mathcal{H}(\mathcal{D}) \subseteq \mathcal{F}(\mathcal{D})$

This theorem is proved for positional GNNs. The demonstration of the other cases follows the same reasoning. Let f and g be respectively the local transition and output functions of the GNN, and consider the following:

$$\begin{aligned} \boldsymbol{x}_n(t+1) &= f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}(t), \boldsymbol{l}_{\mathrm{ne}[n]}), \\ \boldsymbol{o}_n(t+1) &= g(\boldsymbol{x}_n(t+1), \boldsymbol{l}_n), \qquad\qquad n \in \boldsymbol{N}. \end{aligned}$$

where $\boldsymbol{x}_n(0) = \boldsymbol{0}$ holds, for each n. In the following, it is shown by an induction argument on t, that there exists a function $\bar{\kappa}$ such that $\boldsymbol{x}(t) = \bar{\kappa}(\boldsymbol{T}_n^{t+1})$ for $t \geq 1$. Note that this immediately implies that the theorem is true, since we can define a function

$$\kappa(\boldsymbol{T}_n) = \lim_{t \to \infty} g(\bar{\kappa}(\boldsymbol{T}_n^t), \boldsymbol{l}_n) = \lim_{t \to \infty} g(\boldsymbol{x}_n(t-1), \boldsymbol{l}_n) = \varphi(\boldsymbol{G}, n)$$

that satisfies the hypothesis of Theorem 1.

The induction argument goes as follows:

Base: $t = 1$.

The state $\boldsymbol{x}_n(1)$ is computed by applying f on $\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}$. All this data belong to $\boldsymbol{T}_n^2$, so that we can define a function $\bar{\kappa}$ such that

$$\bar{\kappa}(\boldsymbol{T}_n^2) = f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{0}, \boldsymbol{l}_{\mathrm{ne}[n]}) .$$

holds.

Induction: $t > 1$.

Note that $\boldsymbol{x}_n(t)$ is calculated from $\boldsymbol{l}_n, \boldsymbol{x}_{\mathrm{ne}[n]}(t), \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}$. By using the induction argument, there exists $\bar{\kappa}$ such that $\boldsymbol{x}_u(t-1) = \bar{\kappa}(\boldsymbol{T}_u^t)$ holds, for each $u \in \mathrm{ne}[n]$. Thus, $\boldsymbol{x}_n(t)$ depends on $\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}$ and all the $\boldsymbol{T}_u^t$. Since such information is contained in $\boldsymbol{T}_n^{t+1}$, we can define

$$\bar{\kappa}(\boldsymbol{T}_n^{t+1}) = f(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \bar{\kappa}(\boldsymbol{T}_{\mathrm{ne}[n]}^t), \boldsymbol{l}_{\mathrm{ne}[n]}) .$$

where $\bar{\kappa}(\boldsymbol{T}_{\mathrm{ne}[n]}^t)$ is a vector obtained by stacking all the $\bar{\kappa}(\boldsymbol{T}_u^t)$, $u \in \mathrm{ne}[n]$.

VI. Conclusions

In this paper, we studied the approximation properties of graph neural networks, a recently introduced connectionist model for graph processing. First, we defined the class of functions preserving the unfolding equivalence. Such a class contains most of the practically useful maps on graphs. In fact, only when the input graph contains symmetries, the unfolding equivalence may not be preserved. Then, we proved that GNNs can approximate, in probability, up to any degree of precision any function that preserves the unfolding equivalence and that, vice versa, any function implemented by GNNs preserves the unfolding equivalence. The presented results extend and include those already obtained for recursive neural networks, the predecessor model of GNNs, and prove that the GNN model can be applied to more general classes of applications. Some experimental examples shed some light on the computational capability of the model and performances have been discussed w.r.t. the theory developed.

As a topic of future research, it may be useful to consider theoretical issues, e.g. generalizability [29], existence of global minimum [30], of GNNs. For example, the investigation of the generalization properties of GNNs may require the extension of the concepts of Vapnik–Chervonenkis dimension [31] and minimum description length [32]. Moreover, conditions under which the error function does not have any local minima has been considered for feedforward neural networks [33], [34], [30], but not yet for GNNs. Similarly, there are no studies, analogous to those in [35], to the best of our knowledge, on the closure of the class of functions that can be implemented by GNNs.

VII. Acknowledgement

The authors acknowledge financial support from the Australian Research Council in the form of an International Research Exchange scheme which facilitated the visit by the first author to University of Wollongong when the initial work on this paper was performed.

References

[1] S. Haykin, *] Neural Networks: A Comprehensive Foundation*. New York: Prentice Hall, 1994.

[2] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 768–786, September 1998.

[3] M. Hagenbuchner, A. Sperduti, and A. C. Tsoi, "A self-organizing map for adaptive processing of structured data," *IEEE Transactions on Neural Networks*, 2003.

[4] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, vol. 8, pp. 429–459, 1997.

[5] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," Department of Information Engineering, University of Siena, Tech. Rep. DII-1/08, 2007.

[6] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings of the International Joint Conference on Neural Networks*, 2005.

[7] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 3, pp. 303–314, 1989.

[8] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183–192, 1989.

[9] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[10] F. Scarselli and A. C. Tsoi, "Universal approximation using feedforward neural networks: a survey of some existing methods, and some new results," *Neural networks*, pp. 15–37, 1998.

[11] B. Hammer, "Approximation capabilities of folding networks," in *ESANN '99*, Bruges, (Belgium), April 1999, pp. 33–38.

[12] M. Bianchini, M. Maggini, L. Sarti, and F. Scarselli, "Recursive neural networks for processing graphs with labelled edges: Theory and applications," *Neural Networks - Special Issue on Neural Networks and Kernel Methods for Structured Domains*, 2005, to appear.

[13] M. Gori, M. Hagenbuchner, F. Scarselli, and A. C. Tsoi, "Graphical-based learning environment for pattern recognition," in *Structural, Syntactic, and*

Statistical Pattern Recognition: Joint IAPR International Workshops, SSPR 2004 and SPR 2004. Lecture Notes in Computer Science, vol. 3138, 2004, pp. 42–56.

[14] M. A. Khamsi, *An Introduction to Metric Spaces and Fixed Point Theory*. John Wiley & Sons Inc, 2001.

[15] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *Comput. J.*, vol. 7, pp. 155–162, 1964.

[16] L. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," in *IEEE International Conference on Neural Networks*, M. Caudill and C. Butler, Eds., vol. 2. San Diego, 1987: IEEE, New York, 1987, pp. 609–618.

[17] F. Pineda, "Generalization of back–propagation to recurrent neural networks," *Physical Review Letters*, vol. 59, pp. 2229–2232, 1987.

[18] H. A. Priestley, *Introduction to Integration*. Oxford, UK: Oxford University Press, 1997.

[19] Y. Ito, "Differentiable approximation by means of Radon trasformation and its application to neural net," *Journal of computational and applied mathematics*, vol. 55, pp. 31–50, 1994.

[20] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, pp. 551–560, 1990.

[21] K. Hornik, "Approximation capabilities of feedforward neural networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.

[22] M. Bianchini, M. Gori, L. Sarti, and F. Scarselli, "Recursive processing of cyclic graphs," *IEEE Trans. Neural Networks*, 2005, to appear.

[23] V. Di Massa, G. Monfardini, L. Sarti, F. Scarselli, M. Maggini, and M. Gori, "A comparison between recursive neural networks and graph neural networks," in *International Joint Conference on Neural Networks*, July 2006.

[24] G. Monfardini, V. Di Massa, F. Scarselli, and M. Gori, "Graph neural networks for object localization," in *17-th European Conference on Artificial Intelligence*, August 2006.

[25] F. Scarselli, S. Yong, M. Gori, M. Hagenbuchner, A. C. Tsoi, and M. Maggini, "Graph neural networks for ranking web pages," in *Proc. of the 2005 IEEE/WIC/ACM Conference on Web Intelligence*, 2005.

[26] S. Yong, M. Hagenbuchner, F. Scarselli, A. C. Tsoi, and M. Gori, "Document mining using graph neural networks," in *Proceedings of the 5th International Workshop of the Initiative for the Evaluation of XML Retrieval*,, N. Fuhr, M. Lalmas, and A. Trotman, Eds., 2006.

[27] W. Uwents, G. Monfardini, H. Blockeel, F. Scarselli, and M. Gori, "Two connectionist models for graph processing: an experimental comparison on relational data," in *European Conference on Machine Learning*, 2006.

[28] F. Preparata and M. I. Shamos, *Computational geometry: An introduction*. Springer–Verlag, 1985.

[29] W. Maas, "Vapnik–chervonenkis dimension of neural nets," in *The Handbook of Brain Theory and Neural Networks*, M. Arbib, Ed., 1995.

[30] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI–14, no. 1, pp. 76–86, January 1992.

[31] V. N. Vapnik, *Statistical Learning Theory*. Wiley, 1998.

[32] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[33] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural Networks*, vol. 2, pp. 53–58, 1989.

[34] M. Bianchini, M. Gori, and M. Maggini, "On the problem of local minima in recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 167–177, March 1994, special Issue on Recurrent Neural Networks.

[35] M. Gori, F. Scarselli, and A. C. Tsoi, "On the clousure of set of functions that can be realized by a multilayer perceptron," *IEEE Transactions on Neural Networks*, vol. 9, no. 6, pp. 1086–1098, 1998.