Neural Trees for Learning on Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph Neural Networks (GNNs) have emerged as a flexible and powerful approach for learning over graphs. Despite this success, existing GNNs are constrained by their local message-passing architecture and are provably limited in their expressive power. In this work, we propose a new GNN architecture – the *Neural Tree*. The neural tree architecture does not perform message passing on the input graph, but on a tree-structured graph, called the *H-tree*, that is constructed from the input graph. Nodes in the H-tree correspond to subgraphs in the input graph, and they are reorganized in a hierarchical manner such that the parent of a node in the H-tree always corresponds to a larger subgraph in the input graph. We show that the neural tree architecture can approximate any smooth probability distribution function over an undirected graph. We also prove that the number of parameters needed to achieve an ϵ-approximation of the distribution function is exponential in the treewidth of the input graph, but linear in its size. We prove that any continuous $\mathcal{G}$-invariant/equivariant function can be approximated by a nonlinear combination of such probability distribution functions over $\mathcal{G}$. We apply the neural tree to semi-supervised node classification in 3D scene graphs, and show that these theoretical properties translate into significant gains in prediction accuracy, over the more traditional GNN architectures. We also show the applicability of the neural tree architecture to citation networks with large treewidth, by using a graph sub-sampling technique.

1 Introduction

Graph-structured learning problems arise in several disciplines, including biology (*e.g.,* molecule classification [15]), computer vision (*e.g.,* action recognition [19], image classification [48], shape and pose estimation [31]), computer graphics (*e.g.,* mesh and point cloud classification and segmentation [21, 35, 41]), and social networks (*e.g.,* fake news detection [42]), among others [9]. In this landscape, *Graph Neural Networks* (GNN) have gained popularity as a flexible and effective approach for regression and classification over graphs.

Despite this growing research interest, recent work has pointed out several limitations of existing GNN architectures [56, 43, 38, 8]. Local message passing GNNs are no more expressive than the Weisfeiler-Lehman (WL) graph isomorphism test [56], neither can they serve as universal approximators to all $\mathcal{G}$-invariant (equivariant) functions, *i.e.,* functions defined over a graph $\mathcal{G}$ that remain unchanged by (or commute with) node permutation. The work [12] proves an equivalence between the ability to do graph isomorphism testing and the ability to approximate any $\mathcal{G}$-invariant function.

Various GNN architectures have been proposed, that go beyond local message passing or use tensor representations, in order to improve expressivity. Graph isomorphism testing, $\mathcal{G}$-invariant/equivariant

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

function approximation, and the generalized k-order WL (k-WL) tests have served as end objectives and guided recent progress of this inquiry. For example, k-order linear GNN [39] and k-order folklore GNN [38] have expressive powers equivalent to k-WL and $(k+1)$-WL test, respectively [3]. While these architectures can theoretically approximate any $\mathcal{G}$-invariant function (as $k \to \infty$), they use k-order tensors for representations, rendering them impractical for any $k \geq 3$.

There is a need for a new way to look at constructing GNN architectures. With better end objectives to guide theoretical progress. Such an attempt can result in new and expressive GNNs that are provably tractable – if not in general, at least in reasonably constrained settings.

A GNN, by its very definition, operates on graph structured data. The graph structure of the data determines inter-dependency between nodes and their features. Probabilistic graphical models present a reasonable and well-established way of articulating and working with such inter-dependencies in the data. Prior to the advent of neural networks, inference algorithms on such graphical models were successfully applied to many real-world problems. Therefore, we pose that a GNN architecture operating on a graph should have at least the expressive power of a probabilistic graphical model, *i.e.,* it should be able to approximate any distribution defined by a probabilistic graphical model.

This is not a trivial requirement as exact inference (akin to learning the distribution or its marginals) on a probabilistic graphical model, without any structural constraints on the input graph, is known to be an NP-hard problem [13]. Even approximate inference on a probabilistic graphical model is known to be NP-hard in general [46]. A common trick to perform exact inference, consists in constructing a *junction tree* for an input graph and performing message passing on the junction tree instead. In the junction tree, each node corresponds to a subset of nodes of the input graph. The junction tree algorithm remains tractable for graphs with bounded treewidth, while [11] shows that treewidth is the only structural parameter, bounding which, allows for tractable inference on graphical models.

Contribution. We first define the notion of $\mathcal{G}$-compatible function and argue that approximating $\mathcal{G}$-compatible functions is equivalent to approximating any probability distribution on a probabilistic graphical model (Section 4); we also show that $\mathcal{G}$-invariant/equivariant functions considered in related work can be approximated using a nonlinear combination of $\mathcal{G}$-compatible functions.

We then propose a novel GNN architecture – the *Neural Tree* (Section 5). Neural trees do not perform message passing on the input graph, but on a tree-structured graph, called the *H-tree*, that is constructed from the input graph. Each node in the H-tree corresponds to a subgraph of the input graph. These subgraphs are arranged hierarchically in the H-tree such that the parent of a node in the H-tree always corresponds to a larger subgraph in the input graph. The leaf nodes in the H-tree correspond to singleton subsets (*i.e.,* individual nodes) of the input graph. The H-tree is constructed by recursively computing tree decompositions of the input graph and its subgraphs, and attaching them to one another to form a hierarchy. Neural message passing on the H-tree generates representations for all the nodes and important subgraphs of the input graph.

We show that the neural tree architecture can approximate any smooth probability distribution function defined over a given undirected graph (Section 6). We also bound the number of parameters required by a neural tree architecture to obtain an ϵ-approximation of an arbitrary (smooth) distribution function over a graphical model. We show that the number of parameters increases exponentially in the treewidth of the input graph, but only linearly in the input graphs size. Thus, for graphs with bounded treewidth, the neural tree can tractably approximate any smooth distribution function.

We demonstrate the neural tree architecture for semi-supervised node classification in 3D scene graphs and citation networks (Section 7). Our experiments on 3D scene graphs demonstrate that neural trees outperform standard, local message passing GNNs, by a large margin. Citation networks on the other hand, typically have large treewidth; therefore we make use of a recently proposed bounded treewidth graph sub-sampling algorithm [62], that sub-samples the input graph (*i.e.,* removes edges) to reduce its treewidth to a specified number. We show that applying the neural tree architecture in conjunction with such sub-sampling algorithm makes our architecture scalable to large graphs while still preserving its advantage over traditional architectures.

2 Related Work

Expressive Power of Graph Neural Networks. Since the seminal works [18, 50], various GNN architectures have been proposed including Graph Convolutional Networks (GCN) [30, 22, 14, 30, 9], Message Passing Neural Networks (MPNN) [17], GraphSAGE [20], Graph Attention Networks (GAT) [53, 34, 10], message passing GNN [17]. Limited expressive power of these standard GNNs has been a major concern. For instance, it is known that local message passing GNNs can neither distinguish between non-isomorphic graphs (provably worse than the 1-Weisfeiler-Lehman (WL) test) [56, 43], nor can they compute even simple graph properties [16].

Many GNN architectures have been proposed to overcome this expressivity bottleneck. Graph substructure network is proposed in [8] and is shown to be more powerful than the 1-WL test. k-order GNNs, in which message passing is performed among a subset of nodes in the input graph, is shown to have expressive power equivalent to the generalized k-WL test [43, 38]. It is generally understood that to improve the expressivity of GNNs one has to extract features corresponding to important subgraphs, and operate on them. A hierarchical architecture that pools a representation vector from a subset of nodes, at each layer, is proposed in [61]. A junction-tree based message passing GNN is proposed for molecular graph generation in [24].

Graph neural networks have been investigated as function approximators since the beginning. [49] introduces the notion of *unfolding equivalence* and derives a universal approximation result for graph neural networks. Recent research in developing expressive GNN architectures has been towards approximating graph invariant/equivariant functions [40, 39, 38, 28, 47]. While, invariance and equivariance are desirable properties, the problem of designing GNNs that are universal approximators of $\mathcal{G}$-invariant/equivariant functions has been difficult. For instance, the k-order GNNs [39, 38] can provably approximate any graph invariant function, but only as $k \to \infty$, rendering them impractical [3]. An equivalence between designing GNN architectures to approximate graph invariant functions and graph isomorphism testing is shown in [12]. The generalization power of GNNs has also been investigated in [51, 16, 57, 58].

Scene Graphs. Scene graphs are a popular model to abstract information in images or model 3D environments. 2D scene graphs have been used in image retrieval [25], caption generation [27, 1], visual question answering [44, 32], and relationship detection [37]. GNNs are a popular tool for joint object labels and/or relationship inference on scene graphs [55, 36, 59, 63]. Recently, there has been a growing interest towards *3D* scene graphs, which are constructed from 3D data, such as meshes [2], point clouds [54], or raw sensor data [29, 45]. GNNs have been very recently applied to 3D scene graphs for scene layout prediction [54] or object search [33].

3 Problem Statement and Preliminaries

In this section, we state the node classification problem and review standard graph neural networks.

Problem. We focus on the standard problem of *semi-supervised node classification* [30]. We are given a large graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ along with node features $\boldsymbol{X} = (\boldsymbol{x}_v)_{v \in \mathcal{V}}$; where $\boldsymbol{x}_v$ denotes the node feature of node $v \in \mathcal{V}$. The graph is not necessarily connected. A subset of nodes $\mathcal{A} \subset \mathcal{V}$ in $\mathcal{G}$ are labeled, *i.e.,*$\{z_v \in \mathcal{L} \mid v \in \mathcal{A}\}$ is given; here z_v denotes the label for node v and $\mathcal{L}$ the finite set of label classes. We need to design a model to predict the labels of all the unlabeled nodes $v \in \mathcal{V} \backslash \mathcal{A}$.

Graph Neural Networks (GNN). Various GNN architectures have been successfully applied to solve the node classification problem [20, 30, 53, 56, 24]. Standard GNN architectures construct representation vectors for each node in $\mathcal{G}$ by iteratively aggregating representation vectors of its neighboring nodes. At iteration t, the representation vector of node v is generated as follows:

$$h_v^t = \mathrm{AGG}_t \left(h_v^{t-1}, \{ h_u^{t-1} \mid u \in \mathcal{N}_{\mathcal{G}}(v) \} \right), \tag{1}$$

with $h_v^0 \triangleq \boldsymbol{x}_v$, $\forall v \in \mathcal{V}$; where $\mathcal{N}_{\mathcal{G}}(v)$ denotes the set of neighbors of node v in graph $\mathcal{G}$. This process of sharing and aggregating representation vectors among neighboring nodes in $\mathcal{G}$ is often

called *message passing*. This procedure runs for a fixed number of iterations T. The node labels are then generated from the representation vectors at the final iteration T. Node labels are extracted as

$$y_v = \text{READ}(h_v^T), \tag{2}$$

for all $v \in \mathcal{V}$. The functions AGG_t and READ are modeled as single or multi-layer perceptrons.

4 Graph Compatible Functions

We start by defining a class of $\mathcal{G}$-compatible functions. $\mathcal{G}$-compatible functions allow us to establish connections with probabilistic graphical models and the $\mathcal{G}$-invaraint/equivariant functions.

Definition 1 ($\mathcal{G}$-compatible functions) *We say that a function $f : (\times_{v\in\mathcal{V}}\mathbb{X}_v, \mathcal{G}) \to \mathbb{R}$ is compatible with graph $\mathcal{G}$ or $\mathcal{G}$-compatible if it can be factorized as*

$$f(\boldsymbol{X}) = \textstyle\sum_{C\in\mathcal{C}(\mathcal{G})} \theta_C(\boldsymbol{x}_C), \tag{3}$$

where $\mathcal{C}(\mathcal{G})$ denotes the collection of all maximal cliques in $\mathcal{G}$ and θ_C is some function that maps $\times_{v\in C}\mathbb{X}_v$ (the set of node features in the clique C) to a real number.

Compatible functions arise in probabilistic graphical models; for instance, the logarithm of a joint probability distribution is a compatible function (see supplementary material for more examples on how such functions arise in inference on graphical models).

Relation with Invariant/Equivariant Functions. A graph invariant function requires that the function output remains invariant to node permutation, whereas a graph equivariant function outputs a vector (or a tensor in general) which is required to commute with any permutation applied to the input graph nodes. While graph invariance is a desirable property for graph classification problems, graph equivariance is desirable in node classification problems.

We now show that any continuous $\mathcal{G}$-invariant or $\mathcal{G}$-equivariant function can be written as a finite sum of $\mathcal{G}$-compatible functions, each composed with a specific nonlinear function. The precise definitions of $\mathcal{G}$-invariant and $\mathcal{G}$-equivariant functions and the proof is given in the supplementary material.

Theorem 2 (Invariance/Equivariance) *For any continuous $\mathcal{G}$-invariant function $h : \mathbb{X} \to \mathbb{R}$ and an $\epsilon > 0$ there exists an integer $M \geq 1$ and a collection of M continuous $\mathcal{G}$-compatible functions $\{f^i\}_{i=1}^M$ such that*

$$\sup_{\boldsymbol{X}\in\mathbb{X}} \quad \left| h(\boldsymbol{X}) - \textstyle\sum_{i=1}^M \phi\left(f^i(\boldsymbol{X})\right) \right| < \epsilon, \tag{4}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is a nonlinear function. For any continuous $\mathcal{G}$-equivariant function $h : \mathbb{X} \to \mathbb{R}^n$ and an $\epsilon > 0$, (4) *holds for each of its component $h_l(\boldsymbol{X})$.*

The result shows that a GNN architecture that can approximate any $\mathcal{G}$-compatible function will also be able to approximate graph invariant and equivariant functions. In the next section, we describe the neural tree architecture, which can approximate any (smooth) $\mathcal{G}$-compatible function.

5 Neural Tree Architecture

The key idea behind the neural trees architecture is to construct a tree-structured graph from the input graph and perform message passing on the resulting tree instead of the input graph. In the following, we first review the notion of tree decomposition (Section 5.1). We then show how to construct a H-tree for a graph, by successively applying tree decomposition on a given graph $\mathcal{G}$ and its subgraphs (Section 5.2). Finally, we discuss the proposed neural tree architecture for node classification, which performs neural message passing on the H-tree (Section 5.3).

5.1 Tree Decomposition

For a graph $\mathcal{G}$, a *tree decomposition* is a tuple $(\mathcal{T}, \mathcal{B})$ where $\mathcal{T}$ is a tree graph and $\mathcal{B} = \{B_\tau\}_{\tau \in \mathcal{V}(\mathcal{T})}$ is a family of *bags*, where $B_\tau \subset \mathcal{V}(\mathcal{G})$ for every tree node $\tau \in \mathcal{V}(\mathcal{T})$, such that the tuple $(\mathcal{T}, \mathcal{B})$ satisfies the following two properties:

(1) *Connectedness:* for every graph node $v \in \mathcal{V}(\mathcal{G})$, the subgraph of $\mathcal{T}$ induced by tree nodes τ whose bag contains node v, is connected, *i.e.,* $\mathcal{T}_v \triangleq \mathcal{T}\left[\{\tau \in \mathcal{V}(\mathcal{T}) \mid v \in B_\tau\}\right]$ is a connected subgraph of $\mathcal{T}$ for every $v \in \mathcal{V}(\mathcal{G})$.

(2) *Covering:* for every edge $\{u, v\}$ in $\mathcal{G}$ there exists a node $\tau \in \mathcal{V}(\mathcal{T})$ such that $u, v \in B_\tau$.

The simplest tree decomposition of any graph $\mathcal{G}$ is a tree with a single node, whose bag contains all the nodes in $\mathcal{G}$. However, in practical applications, it is desirable to obtain decompositions where the size of the largest bag is small. This is captured by the notion of *treewidth*. The treewidth of a tree decomposition $(\mathcal{T}, \mathcal{B})$ is defined as the size of the largest bag minus one:

$$\text{tw}\left[(\mathcal{T}, \mathcal{B})\right] \triangleq \max_{\tau \in \mathcal{V}(\mathcal{T})} |B_\tau| - 1. \tag{5}$$

The treewidth of a graph $\mathcal{G}$ is defined as the minimum treewidth that can be achieved among all tree decompositions of $\mathcal{G}$. While finding a tree decomposition with minimum treewidth is NP-hard, many algorithms exist that generate tree decompositions with small enough treewidth [4, 5, 52, 6, 7].

We use $(\mathcal{T}, \mathcal{B}) = \texttt{tree-decomposition}(\mathcal{G})$ to denote a generic tree decomposition of a graph $\mathcal{G}$. One of the most popular tree decompositions is the junction tree decomposition, which was introduced in [23]. We denote it by $(\mathcal{T}, \mathcal{B}) = \texttt{junction-tree}(\mathcal{G})$ and describe it's construction in the supplementary material for completeness.

5.2 The H-tree

In this section, we propose and construct a new structure called the H-tree for a given graph $\mathcal{G}$. The H-tree is a tuple $(\mathcal{J}_\mathcal{G}, R)$, where $\mathcal{J}_\mathcal{G}$ is a tree and R is the set of *root* nodes. The key insight behind a H-tree is to capture nodes and subgraphs in the input graph and organize them in a suitable hierarchy – from nodes (as leaves) to large subgraphs (the largest corresponding to the root nodes).

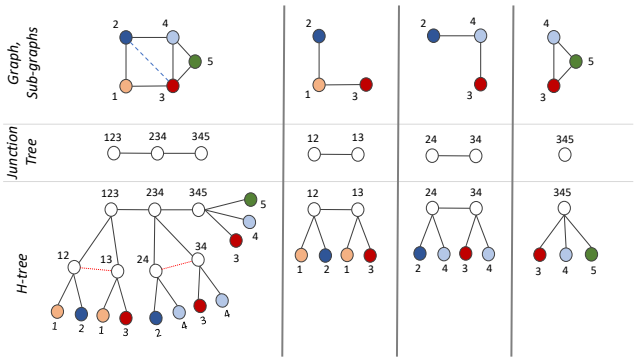

Figure 1: Generating H-trees for graph $\mathcal{G}$ and its subgraphs.

We know that the tree decomposition $(\mathcal{T}, \mathcal{B})$ of a graph captures the important subgraphs (induced by bags $B \in \mathcal{B}$) of the graph. In the H-tree, we further decompose each of these subgraphs with its own tree decomposition, and iterate this process till all the remaining subgraphs (induced by bags) contain only a single node. We start by defining the H-tree of a complete graph.

Definition 3 (Complete graph) *Let $\mathcal{S}_n$ denote a star graph with n leaf nodes and one root. The H-tree $(\mathcal{J}_\mathcal{G}, R)$ of a complete graph $\mathcal{G}$ with n nodes is $\mathcal{J}_\mathcal{G} = \mathcal{S}_n$, where the root node represents the single maximal clique in $\mathcal{G}$ and each of the leaf nodes in $\mathcal{S}_n$ corresponds to a node in $\mathcal{G}$.*

The H-tree for a complete graph of three nodes is shown in Fig. 1, rightmost column. In it, the unique clique in the graph, which contains nodes $\{3, 4, 5\}$, is labeled as $C = (345)$.

The H-tree for a graph that is not complete is constructed recursively as described in Algorithm 1.

Algorithm 1 takes an undirected graph $\mathcal{G}$ and outputs a H-tree $\mathcal{J}_\mathcal{G}$ with a set of root nodes R. To understand the algorithm, let $(\mathcal{T}, \mathcal{B})$ denote a tree decomposition of graph $\mathcal{G}$ (line 1). The H-tree $\mathcal{J}$ is initialized to $\mathcal{J} = \mathcal{T}$ (line 2) and the set of root nodes equals the root nodes of this tree, namely $R = \mathcal{V}(\mathcal{T})$ (line 3). For $B \in \mathcal{B}$, let $\tau(B)$ denote the node corresponding to bag B in $\mathcal{J}$. Then for each bag $B \in \mathcal{B}$ we construct a H-tree of the induced subgraph $\mathcal{G}[B]$ (lines 4-17).

Algorithm 1: `H-tree`

input : Graph $\mathcal{G}$
output : H-tree $(\mathcal{J}_{\mathcal{G}}, R)$

1 $(\mathcal{T}, \mathcal{B}) \leftarrow$ `tree-decomposition`$(\mathcal{G})$
2 $\mathcal{J} \leftarrow \mathcal{T}$
3 $R \leftarrow \mathcal{V}(\mathcal{T})$
4 **for** each bag B in $\mathcal{B}$ **do**
5 **if** $\mathcal{G}[B]$ is a complete graph **then**
6 Update $\mathcal{J}$:
7 $\mathcal{V}(\mathcal{J}) \leftarrow \mathcal{V}(\mathcal{J}) \cup B$
8 $\mathcal{E}(\mathcal{J}) \leftarrow \mathcal{E} \cup \{\{\tau(B), b\}\}_{b \in B}$
9 **else**
10 $(\mathcal{J}', R') \leftarrow$ `H-tree`$(\mathcal{G}[B])$
11 Update $\mathcal{J}$:
12 $\mathcal{E}(\mathcal{J}') \leftarrow \mathcal{E}(\mathcal{J}') \backslash \mathcal{E}(\mathcal{J}'[R'])$
13 $\mathcal{J} \leftarrow \mathcal{J} \cup \mathcal{J}'$
14 $\mathcal{E}(J) \leftarrow \mathcal{E}(J) \cup \{\{\tau(B), r\}\}_{r \in R'}$
15 **end**
16 $\mathcal{J}_G \leftarrow \mathcal{J}$
17 return $(\mathcal{J}_G$, R)
18 **end**

If $\mathcal{G}[B]$ is not complete, we attach its root nodes to $\tau(B)$ (lines 9-15). Specifically, if $(\mathcal{J}', R')$ denotes the H-tree for the induced subgraph $\mathcal{G}[B]$, then we attach the graph $\mathcal{J}'$ to $\mathcal{J}$ by linking all root nodes of $\mathcal{J}'$, namely R', to the node $\tau(B)$ (lines 13-14). To avoid cycles, we also remove edges between the root nodes R' in $\mathcal{J}'$(line 12).

If the induced subgraph $\mathcal{G}[B]$ is complete, then from Definition 3 we know that its H-tree is a star graph with a single clique node, call it C. In this case, we attach the star graph to $\tau(B)$ by merging two nodes – C and $\tau(B)$ – into one. This avoids an unnecessary edge $(\tau(B), C)$ in the H-tree.

Example. Figure 1 shows the construction of a H-tree for a graph with 5 nodes and 6 edges. Here, we have used the `junction-tree` algorithm to perform tree decomposition. The first column shows the graph $\mathcal{G}$ and its junction trees, which has three nodes corresponding to the three cliques in the chordal graph $\mathcal{G}_c$ (which in this case consists in adding the dashed blue line in Figure 1; see supplementary material for details). The remaining columns show the three subgraphs of $\mathcal{G}$ corresponding to each of the three maximal cliques in $\mathcal{G}_c$, along with their junction trees and H-trees. The H-tree of each of these subgraphs is then attached to the junction tree of $\mathcal{G}$ to get the required H-tree for $\mathcal{G}$. The H-tree for graph $\mathcal{G}$ is shown in the last row of the first column in Figure 1. Also illustrated are the two edges deleted (in red) when merging the two H-trees of the subgraphs to the junction tree of $\mathcal{G}$.

Remark 4 (Leaves and features) *Each node in the H-tree (of a graph $\mathcal{G}$) corresponds to a subset of nodes in graph $\mathcal{G}$. Every leaf node l in $\mathcal{J}_{\mathcal{G}}$ corresponds to exactly one node v in $\mathcal{G}$. We denote this node by $\kappa(l)$ for every leaf node l of $\mathcal{J}_{\mathcal{G}}$. In the construction of the H-tree, we also assign the node input feature $\boldsymbol{x}_v$ to every node l in $\mathcal{J}_{\mathcal{G}}$ for which $\kappa(l) = v$. Note that multiple leaf nodes may correspond to a single node v in the graph $\mathcal{G}$,* i.e., *we can have $\kappa(l) = v$ for many leaf nodes l in $\mathcal{J}_{\mathcal{G}}$. Fig. 1 illustrates the input node features by node coloring.*

5.3 Message Passing on the H-tree

Given a graph $\mathcal{G}$ with input node features, we construct a H-tree $(\mathcal{J}_{\mathcal{G}}, R)$ and perform message passing on $\mathcal{J}_{\mathcal{G}}$. We call this the *neural tree* architecture. Representation vectors are generated for each node in the H-tree $\mathcal{J}_{\mathcal{G}}$ by aggregating representation vectors of neighboring nodes in $\mathcal{J}_{\mathcal{G}}$. The message passing starts with $h_l^0 = \boldsymbol{x}_{\kappa(l)}$ for all leaf nodes l in $\mathcal{J}_{\mathcal{G}}$ and $h_u^0 = \mathbf{0}$ for non-leaf nodes u in $\mathcal{J}_{\mathcal{G}}$. These representation vectors are then updated as

$$\boldsymbol{h}_u^t = \mathrm{AGG}_t\left(\boldsymbol{h}_u^{t-1}, \{\boldsymbol{h}_w^{t-1} \mid w \in \mathcal{N}_{\mathcal{J}_{\mathcal{G}}}(u)\}\right), \tag{6}$$

for each iteration $t \in \{1, 2, \ldots T\}$. The aggregation function AGG_t can be modeled in numerous ways. Many of the message passing GNN architectures in the literature, such as GCN [30], GraphSAGE [20], GIN [56], GAT [53, 34], can be used to perform message passing on $\mathcal{J}_{\mathcal{G}}$. After T iterations of message passing, we extract the label y_v for node $v \in \mathcal{G}$ by combining the representation vectors of leaf nodes l of $\mathcal{J}_{\mathcal{G}}$, which correspond to node v in $\mathcal{G}$, *i.e.,* $v = \kappa(l)$:

$$y_v = \mathrm{COMB}\left(\{\boldsymbol{h}_l^T \mid l \text{ leaf node in } \mathcal{J}_{\mathcal{G}} \text{ s.t. } \kappa(l) = v\}\right), \tag{7}$$

for every $v \in \mathcal{V}$, where $\boldsymbol{h}_l^T$ denotes the representation vector generated at leaf node l in $\mathcal{J}_{\mathcal{G}}$ after T iterations. COMB can be modeled by using any of the standard neural network models. In our experiments, we model COMB with a mean pooling function followed by a softmax.

Remark 5 (Mutatis mutandis) *The neural tree architecture is partly inspired by the junction tree algorithm [26]. The junction tree message passing algorithm can be described in three steps. First,*

the clique potentials are computed for all nodes in the junction tree $(\mathcal{T}, \mathcal{B})$ of $\mathcal{G}$. This is followed by message passing between nodes on $\mathcal{T}$, which updates the clique potentials, until convergence. Third, the marginals are computed for each node from the clique potentials. The proposed neural tree can emulate these three steps by message passing from leaf nodes to the root nodes in H-tree, message passing between the root nodes, and message passing back from the root nodes to the leaf nodes, respectively. [57] suggests that such algorithmic alignment of the neural architecture leads to better generalizability. We leave the question of generalization power to future work.

Remark 6 (Scalability and trade-offs) *The proposed architecture requires constructing the H-tree for the graph $\mathcal{G}$, which involves computing a tree decomposition of $\mathcal{G}$. The time and space complexity of computing a tree decomposition of a graph $\mathcal{G}$ scales exponentially in the treewidth of $\mathcal{G}$. In many semi-supervised node classification problems, the treewidth of the input graph is too large to compute a tree decomposition (eg. graphs arising in citation networks [60]). In such cases, to regain computational tractability, one can sub-sample the input graph (*i.e., *remove some edges in $\mathcal{G}$) to get a graph $\mathcal{G}_s$ with smaller treewidth, and then apply the neural tree architecture to this sub-sampled graph $\mathcal{G}_s$. [62] proposes one such graph sub-sampling algorithm, which for any given graph $\mathcal{G}$ and a treewidth bound k, efficiently generates the sub-sampled graph $\mathcal{G}_s$ and its tree decomposition. The complexity of this algorithm is $\mathcal{O}(|\mathcal{E}(\mathcal{G})|(k^2 + |\mathcal{V}(\mathcal{G})|))$. This addition to the neural tree architecture makes it scalable to large graphs (see Section 7.2).*

6 Expressive Power of Neural Trees

We now show that neural trees can learn any graph-compatible function provided it is smooth enough. All proofs are given in the supplementary material.

For simplicity, let the input node features and the representation vectors be real numbers, *i.e.,* $\boldsymbol{x}_v \in \mathbb{R}$ and $\boldsymbol{h}_u^t \in \mathbb{R}$ for all $v \in \mathcal{V}(\mathcal{G})$ and nodes $u \in \mathcal{J}_{\mathcal{G}}$. Let us implement the aggregation function AGG_t in (6) as a shallow network:

$$\boldsymbol{h}_u^t = \text{AGG}_t\left(\boldsymbol{h}_u^{t-1}, \{\boldsymbol{h}_w^{t-1} \mid w \in \mathcal{N}_{\mathcal{J}_{\mathcal{G}}}(u)\}\right) = \texttt{ReLU}\left(\sum_{k=1}^{N_u} a_{u,t}^k \langle \boldsymbol{w}_{u,t}^k, \boldsymbol{h}_{\bar{\mathcal{N}}(u)}^{t-1}\rangle + b_{u,t}^k\right), \quad (8)$$

where $\bar{\mathcal{N}}(u) = \{u\} \cup \mathcal{N}_{\mathcal{J}_{\mathcal{G}}}(u)$ denotes the set containing node u and its neighbors $\mathcal{N}_{\mathcal{J}_{\mathcal{G}}}$.[1] The representation vectors $\boldsymbol{h}_u^t$ are initialized as discussed in Section 5.3. We fix a node v_0 in graph $\mathcal{G}$ and extract our output from v_0:

$$y_{v_0} = \text{COMB}\left(\{\boldsymbol{h}_l^T \mid l \text{ leaf node in } \mathcal{J}_{\mathcal{G}} \text{ s.t. } \kappa(l) = v_0\}\right), \quad (9)$$

where T is the number of iterations. We also assume the COMB function to be a shallow network.

Let N denote the total number of parameters used in the neural tree architecture. Consider the space of functions g that map the input node features $\boldsymbol{X}$ to the output y_{v_0} in (8)-(9):

$$\mathcal{F}(\mathcal{G}, N) = \left\{ g : \boldsymbol{X} \to g(\boldsymbol{X}) = y_{v_0} \;\middle|\; \begin{array}{c}\text{For some } T > 0 \text{ where} \\ y_{v_0} \text{ is given by (8)-(9)}\end{array} \right\}.$$

We now show that any graph-compatible function – that is smooth enough – can be approximated by a function in $\mathcal{F}(\mathcal{G}, N)$ to an arbitrary precision.

Theorem 7 *Let $f : [0,1]^n \to [0,1]$ be a function compatible with a graph $\mathcal{G}$ with n nodes. Let each clique function θ_c in f (see Definition 1) be 1-Lipschitz and be bounded to $[0,1]$. Then, for any $\epsilon > 0$, there exists a $g \in \mathcal{F}(\mathcal{G}, N)$ such that $||f - g||_\infty < \epsilon$, while the number of parameters N is bounded by*

$$N = \mathcal{O}\left(\sum_{u \in \mathcal{V}(\mathcal{J}_{\mathcal{G}})} (d_u - 1)\left(\frac{\epsilon}{d_u - 1}\right)^{-(d_u - 1)}\right), \quad (10)$$

where d_u denotes the degree of node u in $\mathcal{J}_{\mathcal{G}}$, and the summation is over all the non-leaf nodes in $\mathcal{J}_{\mathcal{G}}$.

[1] We assume a different AGG_t function for each node u at iteration t. This choice is more general than our architecture in Section 5.3. However, our results extend to the case where the AGG_t function is the same across nodes u in each iteration t.

We can further develop the bound in Theorem 7 to expose the dependence of the number of parameters N on the treewidth of the tree decomposition of the graph.

Corollary 8 *The number of parameters N in Theorem 7 is upper-bounded by*

$$N = \mathcal{O}\left(n \times (tw\left[\mathcal{J}_{\mathcal{G}}\right] + 1)^{2tw[\mathcal{J}_{\mathcal{G}}]+3} \times \epsilon^{-(tw[\mathcal{J}_{\mathcal{G}}]+1)}\right),$$

where $tw\left[\mathcal{J}_{\mathcal{G}}\right]$ denotes the treewidth of the tree-decomposition of $\mathcal{G}$, formed by the root nodes of $\mathcal{J}_{\mathcal{G}}$.

Remark 9 (Efficient approximations) *Corollary 8 shows that the number of parameters needed to obtain an ϵ-approximation with neural trees increases exponentially in only the treewidth of the tree-decomposition, and is linear in the number of nodes n in the graph. Thus, for graphs with bounded treewidth, neural trees are able to approximate any graph-compatible function efficiently.*

Remark 10 (Data efficiency) *The value of N also affects the data required to train the model: the larger the N, the more samples are required for training. In particular, Corollary 8 provides the reassuring result that if the training dataset contains graphs of small treewidth, then the amount of data required for training scales only linearly in the number of nodes n.*

7 Experiments

7.1 Node Classification in 3D Scene Graphs

We use the neural tree architecture for node classification on 3D scene graphs and show it outperforms the standard GNNs.

Dataset. We run semi-supervised node classification experiments on Stanford's 3D scene graph dataset [2]. The dataset includes 35 3D scene graphs with verified semantic labels, each containing building, room, and object nodes in a residential unit. Since there is only a single class of building nodes (residential), we remove the building node and obtain 482 room-object graphs where each graph contains a room and at least one object in that room as shown in Fig. 2. The resulting dataset has 482 room nodes with 15 semantic labels, and 2338 objects with 35 labels. Each object node is connected to the room node it belongs to. In addition we add 4920 edges to connect adjacent objects in the same room. We use the centroid and bounding box dimensions as features for each node.

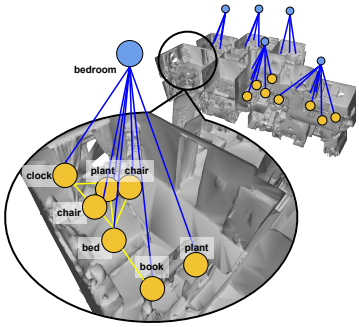

Figure 2: A room-object graph in a 3D scene graph.

Approaches and Setup. We implement the neural tree architecture with four different aggregation functions AGG_t specified in: GCN [30], GraphSAGE [20], GAT [53], GIN [56]. We randomly select 10% of the nodes for validation and 20% for testing. The hyper-parameters of the two approaches are separately tuned based on the best validation accuracy, while using all 70% of the remaining nodes for training; see supplementary material for details.

Results. Table 1 compares the test accuracies (averaged over 100 runs) for the standard GNN architectures and the corresponding neural tree architecture, while using the same type of aggregation function. We see that the neural tree architecture always yields a better prediction model than the standard GNN, for a given aggregation function.

Table 1: Test Accuracy

Model	Input graph	Neural Tree
GCN	40.88 ± 2.28 %	$\mathbf{50.63} \pm 2.25$ %
GraphSAGE	59.54 ± 1.35 %	$\mathbf{63.57} \pm 1.54$ %
GAT	46.56 ± 2.21 %	$\mathbf{62.16} \pm 2.03$ %
GIN	49.25 ± 1.15 %	$\mathbf{63.53} \pm 1.38$ %

To further analyze the proposed architecture, we carry out a series of experiments to see how the test accuracy varies as a function of the amount of training data and the number of message passing iterations T. For simplicity, we only show the neural tree that uses the GCN aggregation function in comparison with the standard GCN.

Figures 3 and 4 plot the test accuracy (averaged over 10 runs) as a function of the training data used and the number of iterations T. The test accuracy –for both the neural tree (NT+GCN) and

GCN– increases with increasing training data, however, the increase is sharper for the neural tree architecture, eventually outperforming GCN.

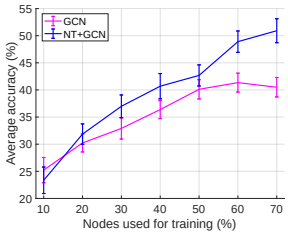

Figure 3: Accuracy vs. training data (% of labeled nodes).

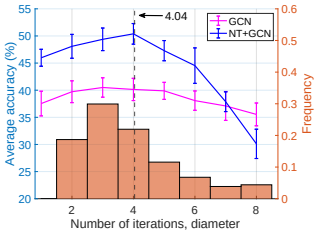

Figure 4: Accuracy vs. number of iterations and weighted diameter distribution.

This shows the higher expressive power of the proposed neural tree architecture. As with the number of iterations T, we see an optimal T at which the test accuracy is maximized. This optimal T is empirically close to the average diameter of the constructed H-trees, of all (room-object) scene graphs in the dataset. This is intuitive, as for the messages to propagate across the entire H-tree, T would have to equal the diameter of the H-tree. See supplementary material for more details, where we also report the compute, train, and test time requirements for neural trees.

7.2 Node Classification in Citation Networks

Here we show that the neural tree architecture can scale to large citation network datasets by using the bounded treewidth graph sub-sampling proposed in [62]. We use the popular citation network datasets [60], where nodes are documents and undirected edges are citations. Each node has a class label representing the subject of the document. These graphs have high treewidth, and therefore, are first sub-sampled using the bounded treewidth graph sub-sampling algorithm in [62]. The neural tree is constructed on the sub-sampled graph.

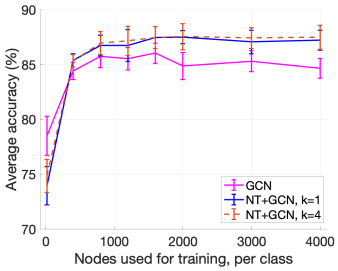

Figure 5: Accuracy vs. training data (per class).

In Figure 5 we plot test accuracy (averaged over 10 runs) as a function of training data, for GCN and NT+GCN, on the Pubmed dataset [60]. As in the scene graph case, we again see that the test accuracy improves with increasing training data, and eventually the neural tree architecture performs better than the standard GNN (GCN in this case). We also observe no significant change in performance of the NT+GCN, with increasing treewidth bound k used for graph sub-sampling. This indicates that it is possible to retain the best possible performance, even after sub-sampling the input graph with a very low treewidth bound. We report the details and more results in the supplementary material.

8 Conclusion

We propose a novel graph neural network architecture – the *neural tree*. The neural tree performs message passing, not on the input graph, but over a constructed H-tree, which provides a tree-structured description of the original graph and its subgraphs. We show that the neural tree architecture can approximate any graph-compatible function, and that the number of parameters required to obtain a desired approximation grows linearly with the number of nodes and exponentially in the treewidth of the input graph; this renders the proposed architecture more parsimonious for large graphs with small treewidth. Graph-compatible functions arise in probabilistic graphical models, hence the proposed architecture can approximate any probability distribution function defined on a graph. Furthermore, we show that a graph-compatible function can be used to approximate any smooth graph-invariant/equivariant functions studied in the literature; this suggests that the goal of approximating graph-compatible functions is a worthwhile pursuit towards the design of novel GNN architectures. We use neural trees for node classification on 3D scene graph and citation network datasets, showing that the proposed architecture leads to more accurate predictions with increasing training data and is applicable even for large networks with high treewidth.

References

[1] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Spice: Semantic propositional image caption evaluation. In *European Conf. on Computer Vision (ECCV)*, pages 382–398, 2016.

[2] Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3D scene graph: A structure for unified semantics, 3D space, and camera. In *Intl. Conf. on Computer Vision (ICCV)*, pages 5664–5673, 2019.

[3] Waiss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. In *Intl. Conf. on Learning Representations (ICLR)*, May 2021.

[4] Ann Becker and Dan Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 81–89, 1996.

[5] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Graph-Theoretic Concepts in Computer Science*, pages 1–14. Springer Berlin Heidelberg, 2006.

[6] Hans L. Bodlaender and Arie M.C.A. Koster. Treewidth computations I: Upper bounds. *Information and Computation*, 208(3):259 – 275, 2010.

[7] Hans L. Bodlaender and Arie M.C.A. Koster. Treewidth computations II: Lower bounds. *Information and Computation*, 209(7):1103 – 1119, 2011.

[8] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, Jan. 2021.

[9] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, 2017.

[10] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Hammerla. Relational graph attention networks. *arXiv preprint arXiv:1904.05811*, Apr. 2019.

[11] Venkat Chandrasekaran, Nathan Srebro, and Prahladh Harsha. Complexity of inference in graphical models. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*, page 70–78, 2008.

[12] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems (NIPS)*, volume 32, Dec. 2019.

[13] G.F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 3844–3852, Dec. 2016.

[15] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 2224–2232, 2015.

[16] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *Intl. Conf. on Machine Learning (ICML)*, volume 119, pages 3419–3430, Jul. 2020.

[17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Intl. Conf. on Machine Learning (ICML)*, volume 70, pages 1263–1272, 2017.

[18] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IEEE Intl. J. Conf. Neural Netw.*, volume 2, pages 729–734, 2005.

[19] M. Guo, E. Chou, D.-A. Huang, S. Song, S. Yeung, and L. Fei-Fei. Neural graph matching networks for fewshot 3D action recognition. In *European Conf. on Computer Vision (ECCV)*, pages 673–689, 2018.

[20] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NIPS)*, page 1025–1035, Dec. 2017.

[21] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. MeshCNN: a network with an edge. *ACM Trans. Graph.*, 38(4):1–12, 2019.

[22] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, Jun. 2015.

[23] Finn V. Jensen and Frank Jensen. Optimal junction trees. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*, page 360–366, Jul. 1994.

[24] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Intl. Conf. on Machine Learning (ICML)*, volume 80, pages 2323–2332, Jul. 2018.

[25] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3668–3678, 2015.

[26] M.I. Jordan. An introduction to probabilistic graphical models. Unpublished Lecture Notes, November 2002.

[27] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[28] Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 32, Dec. 2019.

[29] Ue-Hwan Kim, Jin-Man Park, Taek-Jin Song, and Jong-Hwan Kim. 3-D scene graph: A sparse and semantic representation of physical environments for intelligent agents. *IEEE Trans. Cybern.*, PP:1–13, Aug. 2019.

[30] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Intl. Conf. on Learning Representations (ICLR)*, Apr. 2017.

[31] Nikos Kolotouros, Georgios Pavlakos, and Kostas Daniilidis. Convolutional mesh regression for single-image human shape reconstruction. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[32] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *arXiv preprints arXiv:1602.07332*, 2016.

[33] Andrey Kurenkov, Roberto Martín-Martín, Jeff Ichnowski, Ken Goldberg, and Silvio Savarese. Semantic and geometric modeling with neural message passing in 3D scene graphs for hierarchical mechanical search. *arXiv preprint arXiv:2012.04060*, 2020.

[34] John Boaz Lee, Ryan A. Rossi, Sungchul Kim, Nesreen K. Ahmed, and Eunyee Koh. Attention models in graphs: A survey. *ACM Trans. Knowl. Discov. Data*, 13(6), Nov. 2019.

[35] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs Go as Deep as CNNs? In *Intl. Conf. on Computer Vision (ICCV)*, 2019.

[36] Yikang Li, Wanli Ouyang, Bolei Zhou, Kun Wang, and Xiaogang Wang. Scene graph generation from objects, phrases and region captions. In *International Conference on Computer Vision (ICCV)*, 2017.

[37] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Fei-Fei Li. Visual relationship detection with language priors. In *European Conference on Computer Vision*, pages 852–869, 2016.

[38] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 32, pages 2156–2167, Dec. 2019.

[39] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *Intl. Conf. on Learning Representations (ICLR)*, 2019.

[40] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *Intl. Conf. on Machine Learning (ICML)*, volume 97, pages 4363–4371, Jun. 2019.

[41] Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. Primal-dual mesh convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 33, pages 952–963, Dec. 2020.

[42] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael Bronstein. Fake news detection on social media using geometric deep learning. *arXiv 1902.06673*, 2019.

[43] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *Nat. Conf. on Artificial Intelligence (AAAI)*, 33(01):4602–4609, Jul. 2019.

[44] Mengye Ren, Ryan Kiros, and Richard S. Zemel. Image question answering: A visual semantic embedding model and a new dataset. *arXiv preprints arXiv:1505.02074*, 2015.

[45] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone. 3D dynamic scene graphs: Actionable spatial perception with places, objects, and humans. In *Robotics: Science and Systems (RSS)*, 2020.

[46] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, Apr. 1996.

[47] Akiyoshi Sannai, Yuuki Takai, and Matthieu Cordonnier. Universal approximations of permutation invariant/equivariant functions by deep neural networks. *arXiv preprint arXiv:1903.01939*, Sep. 2019.

[48] Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *Intl. Conf. on Learning Representations (ICLR)*, 2018.

[49] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20(1):81–102, 2009.

[50] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Netw.*, 20(1):61–80, 2008.

[51] F. Scarsellia, A. Tsoi, and M. Hagenbuchner. The Vapnik–Chervonenkis dimension of graph and recursive neural networks. *Neural Networks*, 108:248–259, 2018.

[52] Alun Thomas and Peter J. Green. Enumerating the junction trees of a decomposable graph. *J. Comput Graph Stat.*, 18(4):930–940, Dec. 2009.

[53] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lió, and Yoshua Bengio. Graph attention networks. In *Intl. Conf. on Learning Representations (ICLR)*, May 2018.

[54] Johanna Wald, Helisa Dhamo, Nassir Navab, and Federico Tombari. Learning 3D semantic scene graphs from 3D indoor reconstructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3961–3970, 2020.

[55] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3097–3106, 2017.

[56] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Intl. Conf. on Learning Representations (ICLR)*, May 2019.

[57] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *Intl. Conf. on Learning Representations (ICLR)*, 2020.

[58] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon Shaolei Du, Ken-Ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. In *Intl. Conf. on Machine Learning (ICML)*, 2021.

[59] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph R-CNN for scene graph generation. In *European Conf. on Computer Vision (ECCV)*, 2018.

[60] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Intl. Conf. on Machine Learning (ICML)*, page 40–48, Jun. 2016.

[61] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4800–4810, 2018.

[62] Jaemin Yoo, U Kang, Mauro Scanagatta, Giorgio Corani, and Marco Zaffalon. Sampling subgraphs with guaranteed treewidth for accurate and efficient graphical inference. In *Int. Conf. on Web Search and Data Mining*, page 708–716, Jan. 2020.

[63] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural motifs: Scene graph parsing with global context. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   [Yes]
   (b) Did you describe the limitations of your work?
   [Yes]
   (c) Did you discuss any potential negative societal impacts of your work?
   [No] There aren't any direct negative societal impact of this work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them?
   [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results?
   [Yes]
   (b) Did you include complete proofs of all theoretical results?
   [Yes] . The detailed proofs of all our results are given in the supplementary material, due to page limitation on the paper.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)?
   [Yes] . We provide all the information needed (in the paper and in the supplementary material) to reproduce all the experimental results in the paper.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)?
   [Yes] . Yes, we provide all the training details: the train/val/test split, the procedure we use to tune hyperparameters, and also the list of optimized hyperparameetrs in each case. Most of these can be found in the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)?
   [Yes] . All our prediction test accuracy plots show error bars, after running experiments multiple times with random seed.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)?
   [Yes] . We report the timing requirements in the supplementary material due to space limitation in the main paper.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators?
   [Yes] We only use two existing datasets and we have cited the creators.
   (b) Did you mention the license of the assets?
   [No] We only use two existing datasets and we have cited the creators. These datasets are in the public domain and are available for use under the MIT License.
   (c) Did you include any new assets either in the supplemental material or as a URL?
   [No] We only use and work with the existing dataset that are in the public domain and are available for use under the MIT License.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating?
   [No] We only use existing datasets that are in the public domain and are available for use under the MIT License.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content?

[No] The data we are using does not contain any personally identifiable information or offensive content.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable?

[No] . We did not use crowdsourcing nor did we conduct any research with human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable?

[No] . We did not use crowdsourcing nor did we conduct any research with human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation?

[No] . We did not use crowdsourcing nor did we conduct any research with human subjects.