THEORY OF GRAPH NEURAL NETWORKS: REPRESENTATION AND LEARNING

STEFANIE JEGELKA

ABSTRACT

Graph Neural Networks (GNNs), neural network architectures targeted to learning representations of graphs, have become a popular learning model for prediction tasks on nodes, graphs and configurations of points, with wide success in practice. This article summarizes a selection of emerging theoretical results on approximation and learning properties of widely used message passing GNNs and higher-order GNNs, focusing on representation, generalization, and extrapolation. Along the way, it summarizes broad mathematical connections.

MATHEMATICS SUBJECT CLASSIFICATION 2020

Primary 68T05; Secondary 05C62, 05C60, 05C90, 68Q32, 68R10

KEYWORDS

Machine learning, graph representation learning, graph isomorphism, approximation theory, learning theory

ICM INTERNATIONAL CONGRESS
OF MATHEMATICIANS
2022 JULY 6—14

Proc. Int. Cong. Math. 2022, Vol. 7, pp. 5450–5476
DOI 10.4171/ICM2022/162

Published by EMS Press

1. INTRODUCTION

There has been growing interest in solving machine learning tasks when the input data is given in form of a graph $G = (V, E, X, W)$ from a set of attributed graphs $\mathcal{G}$, where $X \in \mathbb{R}^{d \times |V|}$ contains vectorial attributes for each node, and $W \in \mathbb{R}^{d_w \times |E|}$ contains attributes for each edge (X and W may be empty). Examples include predictions in social networks, recommender systems and link prediction (given two nodes, predict an edge), property prediction of molecules, prediction of drug interactions, traffic prediction, forecasting physics simulations, and learning combinatorial optimization algorithms for hard problems. These examples use two types of task: (1) given a graph G, predict a label $F(G)$; (2) given a graph G and node $v \in V(G)$, predict a node label $f(v)$. An edge may be similarly predicted, but from two nodes instead of one.

Solving these tasks demands a sufficiently rich *embedding* of the graph or each node that captures structural properties as well as the attribute information. While graph embeddings have been a widely studied topic, including spectral embeddings and graph kernels, recently, *Graph Neural Networks (GNNs)* [36,37,39,49,65,83] have emerged as an empirically broadly successful model class that, as opposed to, e.g., spectral embeddings, allows adapting the embedding to the task at hand, generalizes to other graphs of the same input type, and incorporates attributes. Due to space limits, this survey focuses on the popular message passing (spatial) GNNs, formally defined below, and their rich mathematical connections, with an excursion into higher-order GNNs.

When *learning* a GNN, we observe N i.i.d. samples $\mathcal{D} = \{G_i, y_i\}_{i=1}^N \in (\mathcal{G} \times \mathcal{Y})^N$ drawn from an underlying distribution $\mathcal{P}$ on $\mathcal{G} \times \mathcal{Y}$. The *labels* y_i are often given by an unknown *target function* $g(G_i)$, and observed with or without i.i.d. noise. Given a (convex) loss function $\ell : \mathcal{G} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ that measures prediction error, i.e., mismatch of y and $F(G)$, such as the squared loss or cross-entropy, we aim to estimate a model F from our GNN model class $\mathcal{F}$ to minimize the expected loss (*population risk*) $\mathcal{R}(F)$:

$$\min_{F \in \mathcal{F}} \mathbb{E}_{(G,y) \sim \mathcal{P}} \big[\ell\big(G, y, F(G)\big)\big] \equiv \min_{F \in \mathcal{F}} \mathcal{R}(F). \tag{1.1}$$

When analyzing this quantity, three main questions become important:

1. Representational power (Section 2). Which target functions g can be approximated well by a GNN model class $\mathcal{F}$? Answers to this question relate to graph isomorphism testing, approximation theory for neural networks, local algorithms and representing invariance/equivariance under permutations.

2. Generalization (Section 3). Even with sufficient approximation power, we can only estimate a function $\hat{F} \in \mathcal{F}$ from the data sample $\mathcal{D}$. The common learning or *training* procedure is to instead minimize the *empirical risk* $\widehat{\mathcal{R}}(F)$:

$$\hat{F} \in \arg\min_{F \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell\big(G_i, y_i, F(G_i)\big) \equiv \arg\min_{F \in \mathcal{F}} \widehat{\mathcal{R}}(F). \tag{1.2}$$

Generalization asks how well $\hat{F}$ is performing according to the population risk, i.e., $\widehat{\mathcal{R}}(\hat{F})$, as a function of N and model properties. Good generalization may demand explicit (e.g., via penalties) or implicit regularization (e.g., via the optimization algorithm, typically variants of stochastic gradient descent). Hence, generalization analyses involve the complexity of the model class $\mathcal{F}$, the target function, the data and the optimization procedure.

3. Generalization under distribution shifts (Section 4). In practice, a learned model $\hat{F}$ is often deployed on data from a distribution $\mathcal{Q} \neq \mathcal{P}$, e.g., graphs of different size, degree or attribute ranges so that for instance $\text{supp}(Q) \supset \text{supp}(P)$. In which cases can we expect successful *extrapolation* to $\mathcal{Q}$? This depends on the structure of the graphs and the task, formalizable via graph limits, local structures and algorithmic structures, e.g., dynamic programming.

Beyond these topics, GNNs have close connections to graph signal processing as learnable filters, to geometric learning and probabilistic inference.

1.1. Graph Neural Networks (GNNs)

Message passing graph neural networks (MPNNs) follow an iterative scheme [36, 37, 39, 49, 65, 83]. Throughout, they maintain a representation (embedding) $h_v^{(t)} \in \mathbb{R}^{d_t}$ for each node $v \in V$. In each iteration t, we update each node v's embedding $h_v^{(k)}$ as a function of its neighbors' embeddings and possible edge attributes:

$$h_v^{(0)} = x_v, \quad \forall v \in V, \tag{1.3}$$

$$m_v^{(t)} = f_{\text{Agg}}^{(t)}\big(h_v^{(t-1)}, \{\{h_u^{(t-1)}, w(u,v) \mid u \in \mathcal{N}(v)\}\}\big), \quad 1 \le t < T \quad \text{(Aggregate)}, \tag{1.4}$$

$$h_v^{(t)} = f_{\text{Up}}\big(h_v^{(t)}, m_v^{(t)}\big) \quad \text{(Update)}. \tag{1.5}$$

The final node representation $f(v) = h_v^{(T)}$, $\forall v \in V$ is the last iterate, possibly concatenated with a linear classifier. Here, $\mathcal{N}(v) \subset V$ denotes the neighborhood of $v \in V$, and $\{\{\cdot\}\}$ a multiset. Via the updates, $h_v^{(t)}$ encodes the t-hop neighborhood of node v, i.e., the subgraph of all nodes reachable from v within t steps. The number of iterations T is also termed the GNN *depth*, and one iteration may be viewed as a layer.

The *aggregation function* $f_{\text{Agg}}^{(t)} : \mathbb{R}^{d_{t-1}} \to \mathbb{R}^{d_t}$ plays a major role and is shared by all nodes within an iteration. It is a nonlinear function of the form

$$f_{\text{Agg}}^{(t)}\big(h_v^{(t-1)}, \{\{h_u^{(t-1)}, w(u,v) \mid u \in \mathcal{N}(v)\}\}\big) = \phi_1^{(t)}\Big(\sum_{u \in \mathcal{N}(v)} \phi_2^{(t)}\big(h_u^{(t)}, h_v^{(t)}, w(u,v)\big)\Big). \tag{1.6}$$

The sum may also be replaced by an average, degree-normalized sum or coordinate-wise min or max. In the most general form, the functions ϕ_1, ϕ_2 are implemented as *multilayer perceptrons (MLPs)*, neural networks that alternate linear transformations and coordinate-wise nonlinear activations such as the ReLU ($\sigma(a) = \max\{a, 0\}$) or sigmoid function:

$$\text{MLP}(h; \theta) = \sigma\big(W^{(M)} \cdots \sigma\big(W^{(2)}\sigma(W^{(1)}h + b^{(1)}) + b^{(2)}\big) \cdots + b^{(M)}\big). \tag{1.7}$$

The learnable parameters θ of the MLP are the weight matrices $W^{(j)}$ and bias vectors $b^{(j)}$. The update f_{Up} is typically a weighted combination with learnable weight matrices:

$$f_{\text{Up}}\big(h_v^{(t)}, m_v^{(t)}\big) = \sigma\big(W_1^{(t)}h_v^{(t)} + W_2^{(t)}m_v^{(t)}\big) \quad \text{or} \quad f_{\text{Up}}\big(h_v^{(t)}, m_v^{(t)}\big) = m_v^{(t)}. \tag{1.8}$$

Finally, if a graph-level prediction is desired, all node representations can be aggregated by a permutation invariant *readout* function

$$F(G) = f_{\text{Read}}\big(\{\{h_v^{(T)} \mid v \in V\}\}\big). \tag{1.9}$$

Here, we assume the readout has the form (1.6) or is a simple sum or average. Typically, all parameters are learned jointly via stochastic gradient descent minimizing the empirical risk.

Throughout this article, $n = |V|$ denotes the number of nodes and N the number of training data points.

Permutation invariance. An important property of GNNs is permutation invariance of the graph, and equivariance of the node representations. Let $A \in \mathbb{R}^{n \times n}$ be the adjacency matrix of a graph $G \in \mathcal{G}$, and $X \in \mathbb{R}^{n \times d}$ its node features. Permutation invariance/equivariance means that for all permutation matrices $P \in \mathbb{R}^{n \times n}$ and all $G \in \mathcal{G}$:

$$F(PAP^\top, PX) = F(A, X) \tag{1.10}$$

$$f(PAP^\top, PX, v) = f(A, X, v) \tag{1.11}$$

2. REPRESENTATIONAL POWER OF GNNS

For functions on graphs, representational power has mainly been studied in terms of graph isomorphism: which graphs a GNN can distinguish. Via variations of the Stone–Weierstrass theorem, these results yield universal approximation results. Other works bound the ability of GNNs to compute specific polynomials of the adjacency matrix and to distinguish graphons [28, 60]. Observed limitations of MPNNs have inspired higher-order GNNs (Section 2.3). Moreover, if all node attributes are unique, then analogies to local algorithms yield algorithmic approximation results and lower bounds (Section 2.2).

2.1. GNNs and graph isomorphism testing

A standard characterization of the discriminative power of GNNs is via the hierarchy of the *Weisfeiler–Leman (WL)* algorithm for graph isomorphism testing, also known as color refinement or vertex classification [75], which was inspired by the work of Weisfeiler and Leman [93, 94]. The WL algorithm does not entirely solve the graph isomorphism problem, but its power has been widely studied.

A *labeled* graph is a graph endowed with a node coloring $l : V(G) \to \Sigma$ for some sufficiently large alphabet Σ. Given a labeled graph (G, l), the 1-dimensional WL algorithm (1-WL) iteratively computes a node coloring $c_l^{(t)} : V(G) \to \Sigma$ for some sufficiently large alphabet Σ. Starting with $c_l^{(0)}$ in iteration $t = 0$, in iteration $t > 0$ it sets for all $v \in V$,

$$c_l^{(t)}(v) = \text{Hash}\big(c_l^{t-1}(v), \{\{c_l^{t-1}(u) \mid u \in \mathcal{N}(v)\}\}\big), \tag{2.1}$$

where Hash is an injective map from the input pair to Σ, i.e., it assigns a unique color to each neighborhood pattern. To compare two graphs G, G', the algorithm compares the multisets $\{\{c_l^{(t)}(v) \mid v \in V(G)\}\}$ and $\{\{c_l^{(t)}(u) \mid u \in V(G')\}\}$ in each iteration. If the sets differ, then it determines that $G \neq G'$. Otherwise, it terminates when the number of colors in iteration t and $t-1$ are the same, which occurs after at most $\max\{|V(G)|, |V(G')|\}$ iterations.

The computational analogy between the 1-WL algorithm and MPNNs is obvious. Since the WL algorithm uniquely colors each neighborhood, the coloring $c_l^{(t)}(v)$ always *refines* the coloring $h_v^{(t)}$ from a GNN.

Theorem 1 ([66, 95]). *If for two graphs G, G' a message passing GNN outputs $f_G(G) \neq f_G(G')$, then the 1-WL algorithm will determine that $G \neq G'$.*

For any t, there exists an MPNN such that $c_l^{(t)} \equiv h^{(t)}$. A sufficient condition is that the aggregate, update, and readout operations are injective multiset functions.

GNNs that use the degree for normalization in the aggregation [49] can be equivalent to the 1-WL agorithm too, but with one more iteration in the WL algorithm [35].

2.1.1. Representing multiset functions

Theorem 1 demands the neighbor aggregation f_{Agg} to be an injective multiset function. Theorem 2 shows how to universally approximate multiset functions.

Theorem 2 ([92, 95]). *Any multiset function G on a countable domain can be expressed as*

$$G(S) = \phi_1\Big(\sum\nolimits_{s \in S} \phi_2(s)\Big), \tag{2.2}$$

where $\phi_1 : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ and $\phi_2 : \mathbb{R}^{d_2} \to \mathbb{R}$ are nonlinear functions.

The proof idea is to show that there exists an injective function $\sum_{s \in S} \phi(s)$. The above result is an extension of a universal approximation result for set functions [72, 73, 101], and suggests a neural network model for sets where ϕ_1, ϕ_2 are approximated by MLPs. The Graph Isomorphism Network (GIN) [95] implements this sum decomposition in the aggregation function to ensure the ability to express injective operations.

Here, the latent dimension d_2 plays a role. Proofs for countable domains use a discontinuous mapping ϕ_1 into a fixed-dimensional space, whereas MLPs universally approximate *continuous* functions [25]. Continuous set functions on $\mathbb{R}^{\leq M}$ (i.e., $|S| \leq M$) can be sum-decomposed as above with continuous ϕ_1, ϕ_2 and latent dimension at least $d_2 = M$. The dimension is a necessary and sufficient condition for universal approximation [92]. For GNNs, this means d_2 must be at least the maximum degree $\deg(G)$ of the input graph G.

2.1.2. Implications for graph distinction

Theorem 1 allows directly transferring any known result for 1-WL to MPNNs. For instance, 1-WL succeeds in distinguishing graphs sampled uniformly from all graphs on n nodes with high probability, and failure probability going to zero as $n \to \infty$ [8, 9]. 1-WL can also distinguish any nonisomorphic pair of trees [42]. It fails for regular graphs, as all node colors will be the same. The graphs that 1-WL can distinguish from any nonisomorphic graph can be recognized in quasilinear time [6]. See also [6, 18, 48] for more detailed results on the expressive power of variants of the WL algorithm.

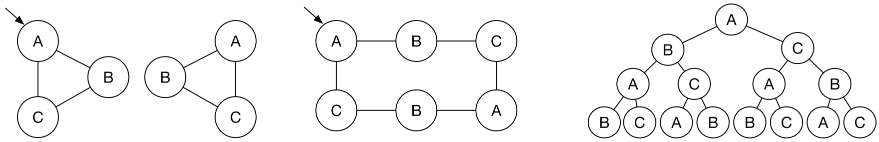

FIGURE 1

Graphs G_1 (left, 2 connected components) and G_2 (middle) with node attributes indicated by letters. The computation tree rooted at the node with arrow (right) agrees in both graphs, and likewise for the other nodes. Hence, 1-WL and MPNNs cannot distinguish G_1 and G_2. Figure adapted from [33].

2.1.3. Computation trees and structural graph properties

To further illustrate the implications of GNNs' discriminative power, we look at some specific examples. The maximum information contained in any embedding $h_v^{(t)}$ can be characterized by a computation tree $\mathcal{T}(h_v^{(t)})$, i.e., an "unrolling" of the message passing procedure. 1-WL essentially colors computation trees. The tree $\mathcal{T}(h_v^{(t)})$ is constructed recursively: let $\mathcal{T}(h_v^{(0)}) = x_v$ for all $v \in V$. For $t > 0$, construct a root with label x_v and, for any $u \in \mathcal{N}(v)$ construct a child subtree $\mathcal{T}(h_u^{(t-1)})$. Figure 1 illustrates an example.

Proposition 1. *If for two nodes $u \neq v$, we have $\mathcal{T}(h_v^{(t)}) = \mathcal{T}(h_u^{(t)})$, then $h_v^{(t)} = h_u^{(t)}$.*

Comparing computation trees directly implies that MPNNs cannot distinguish regular graphs. It also shows further limitations with practical impact (Fig. 1), in particular for learning combinatorial algorithms, and for predicting properties of molecules, where functional groups are of key importance. We say a class of models $\mathcal{F}$ *decides* a graph property if there exists an $F \in \mathcal{F}$ such that for any two G, G' that differ in the property, we obtain $F(G) \neq F(G')$.

Proposition 2. *MPNNs cannot decide girth, circumference, diameter, radius, existence of a conjoint cycle, total number of cycles, and existence of a k-clique* [33]. *MPNNs cannot count induced (attributed) subgraphs for any connected pattern of 3 or more nodes, except star-shaped patterns* [22].

Motivated by these limitations, generalizations of GNNs were proposed that provably increase their representational power. Two main directions are to (1) introduce node IDs (Section 2.2), and (2) use higher-order functions that act on tuples of nodes (Section 2.3).

2.2. Node IDs, local algorithms, combinatorial optimization, and lower bounds

The major weaknesses of MPNNs arise from their inability to identify nodes as the origin of specific messages. Hence, MPNNs can be strengthened by making nodes more distinguishable. The gained representational power follows from connections with local algorithms, where the input graph defines both the computational problem and the network topology of a distributed system: each node $v \in V$ is a local machine and generates a local output, and all nodes execute the same algorithm, without faults.

Approximation algorithms. Sato et al. [80] achieve a partial node distinction by transferring the idea of a *port numbering* from local algorithms. Edges incident to each node are numbered as outgoing ports. In each round, each node simultaneously sends a message to each port, but the messages can differ across ports:

$$m_v^{(t)} = f_{\mathrm{Agg}}^{(t)}\big(\{\!\{(\mathrm{port}(u,v), \mathrm{port}(v,u), h_u^{(t-1)}) \mid u \in \mathcal{N}(v)\}\!\}\big). \tag{2.3}$$

Permutation invariance, though, is not immediate. This corresponds to the vector–vector consistent (VV_C) model for local algorithms [41]. The VV_C analogy allows transferring results on representing approximation algorithms. CPNGNN is a specific VV_C GNN model.

Theorem 3 ([80]). *There exists a CPNGNN that can compute a* $(\deg(G)+1)$*-approximation for the minimum dominating set problem, a CPNGNN that can compute a 2-approximation for the minimum vertex cover problem, but no CPNGNN can do better. No CPNGNN can compute a constant-factor approximation for the maximum matching problem.*

Adding a weak vertex 2-coloring leads to further results. Despite the increased power compared to MPNN, CPNGNNs retain most limitations of Proposition 2 [33].

A more powerful alternative is to endow nodes with fully unique identifiers [59,81]. For example, augmenting the GIN model (a maximally expressive MPNN) [95] with random node identifiers yields a model that can decide subgraphs that MPNN and CPNGNN cannot [81]. This model can further achieve better approximation results for minimum dominating set ($H(\deg(G)+1)+\varepsilon$), where H is the harmonic number) and maximum matching ($1+\varepsilon$).

Turing completeness. Analogies to local algorithms imply that MPNNs with unique node IDs are *Turing complete*, i.e., they can compute any function that a Turing machine can compute, including graph isomorphism. In particular, the proof shows an equivalence to the Turing universal LOCAL model from distributed computing [3,57,69].

Theorem 4 ([59]). *If f_{Up} and f_{Agg} are Turing complete functions and the GNN gets unique node IDs, then GNN and LOCAL are equivalent. For any MPNN F there exists a local algorithm $\mathcal{A}$ of the same depth, such that $F(G) = \mathcal{A}(G)$, and vice versa.*

Corollary 1 ([59]). *Under the conditions in Theorem 4, if the GNN depth (number of iterations) is at least* $\mathrm{diameter}(G)$ *and the width is unbounded, then MPNNs can compute any Turing computable function over connected attributed graphs.*

Lower bounds. The *width* of a GNN refers to the dimensionality of the embeddings $h_v^{(t)}$. For bounded size, GNNs lose computational power. Via analogies to the CONGEST model [70], which bounds message sizes, one can transfer results on decision, optimization and estimation problems on graphs. These lead to lower bounds on the product of depth and width of the GNN. Here, the nodes do not have access to a random generator.

Theorem 5 ([59]). *If a problem cannot be solved in less than T rounds in CONGEST using messages of at most b bits, then it cannot be solved by an MPNN of width $w \leq (b - \log_2 n)/p = O(b/\log n)$ and depth T, where $p = \Theta(n)$.*

Theorem 5 directly implies lower bounds for solving combinatorial problems, e.g., $Tw = \Omega(n/\log n)$ for cycle detection and computing diameter, and $T\sqrt{w} = \Omega(\sqrt{n}/\log n)$ for minimum spanning tree, minimum cut, and shortest path [59].

Moreover, we can transfer ideas from communication complexity. The *communication capacity* c_f of an MPNN f (with unique node IDs) is the maximum number of symbols that the MPNN can transmit between any two disjoint sets $V_1, V_2 \subset V$ of nodes when viewed as a communication network: $c_f \leq \mathrm{cut}(V_1, V_2) \sum_{t=1}^{T} \min\{m_t, w_t\} + \sum_{t=1}^{T} \gamma_t$, where T is the GNN depth, w_t the width of layer t, m_t the size of the messages, and γ_t the size of a global state that is maintained. The communication capacity of the MPNN must be at least $c_f = \Omega(n)$ to distinguish all trees, and $c_f = \Omega(n^2)$ to distinguish all graphs [58]. By relating discrimination and function approximation (Section 2.4), these results have implications for function approximation, too.

Random node IDs. While unique node IDs are powerful in theory, in many practical examples the input graphs do not have unique IDs. An alternative is to assign random node IDs [1, 27]. This can still yield GNNs that are essentially permutation invariant: while their outputs are random, the outputs for different graphs are still sufficiently separated [1]. This leads to a probabilistic universal approximation result:

Theorem 6 ([1]). *Let $h : \mathcal{G} \to \mathbb{R}$ be a permutation invariant function on graphs of size $n \geq 1$. Then for all $\varepsilon, \delta > 0$ there exists an MPNN F with access to a global readout and with random node IDs such that for every $G \in \mathcal{G}$ it holds that* $\Pr(|F(G) - h(G)| \leq \varepsilon) \geq 1 - \delta$.

The proof builds on a result by [10] that states that any logical sentence in FOC_2 can be expressed by the addressed GNN. The logic considered here is a fragment of first-order (FO) predicate logic that allows to incorporate counting quantifiers of the form $\exists^{\geq k} x \psi(x)$, i.e., there are at least k elements x satisfying ψ, but is restricted to two variables. FOC_2 is tightly linked with the 1-WL test: for any nodes $u, v \in V$ in any graph, 1-WL colors u and v the same if and only if they are classified the same by all FOC_2 classifiers [19].

2.3. Higher-order GNNs

Instead of adding unique node IDs, one may increase the expressive power of GNNs by encoding subsets of V that are larger than the single nodes used in MPNNs. Three such directions are: (1) neural network versions of higher-dimensional WL algorithms, (2) (non)linear equivariant operations, and (3) recursion. Other strategies that could not be covered here use, e.g., simplicial and cell complexes [16, 17] or augment node attributes with topological information (e.g., persistent homology) [102].

Most of these GNNs act on k-tuples $s \in V^k$, and may be written in a unified form via tensors $H^{(t)} \in R^{n^k \times d_t}$, where the first k coordinates index the tuple, and $H^{(t)}_{s,:} \in \mathbb{R}^{d_t}$ is the representation of tuple s in layer t. For MPNNs, which use node and edge information, $H^{(0)} \in \mathbb{R}^{n \times n \times (d+1)}$. The first d channels of $H^{(0)}$ encode the node attributes: $H^{(0)}_{v,v,1:d} = x_v$ and $H^{(0)}_{u,v,1:d} = 0$ for $u \neq v$. The final channel captures the adjacency matrix A of the graph:

$H^{(0)}_{:,:,(d+1)} = A$. Node embeddings are computed by a permutation equivariant network:

$$f(G) = m \circ S_E \circ F^{(T)} \circ \cdots \circ F^{(1)} \circ \text{SHAPE}(G), \tag{2.4}$$

where $m : \mathbb{R}^{d_T} \to \mathbb{R}^{d_{\text{out}}}$ is an MLP that is applied to each representation h_v^T separately, $S_E : \mathbb{R}^{n^k \times d_T} \to \mathbb{R}^{n \times d_T}$ is a reduction $S_E(H)_{v,:} = \sum_{s \in V^k : s_1 = v} H_{s,:}$, and each layer $F^{(t)} : \mathbb{R}^{n^k \times d_{t-1}} \to \mathbb{R}^{n^k \times d_t}$ is a message passing (aggregation and update) operation for MPNNs, and will be defined for higher-order networks. The first operation shapes the input into the correct tensor form, if needed. For a graph embedding, we switch to a reduction $S_I : \mathbb{R}^{n^k \times d_T} \to \mathbb{R}^{d_T}$, $S_I(H) = \sum_{s \in V^k} H_{s,:}$ and apply the MLP m to the resulting vector: $F(G) = m \circ S_I \circ F^{(T)} \circ \cdots \circ F^{(1)} \circ \text{SHAPE}(\text{G})$. The GNNs differ in their layers $F^{(t)}$.

2.3.1. Higher-order WL networks

Extending analogies of MPNNs and the 1-WL algorithm [66, 95], the first class of higher-order GNNs imitates versions of the k-dimensional WL algorithm. The k-WL algorithms are defined on k-tuples of nodes, and different versions differ in their aggregation and definition of neighborhood. In iteration 0, the *k-WL* algorithm labels each k-tuple $s \in V^k$ by a unique ID for its isomorphism type. Then it aggregates over neighborhoods $\mathcal{N}_i^{\text{WL}}(s) = \{(s_1, s_2, \ldots, s_{i-1}, v, s_{i+1}, \ldots, s_k) \mid \forall v \in V\}$ for $1 \leq i \leq k$:

$$c_i^{(t)}(s) = \{\!\{c^{(t-1)}(s') \mid s' \in \mathcal{N}_i^{\text{WL}}(s)\}\!\}, \quad 1 \leq i \leq k,\ s \in V^k, \tag{2.5}$$

$$c^{(t)}(s) = \text{Hash}\big(c^{(t-1)}(s), c_1^{(t)}(s), c_2^{(t)}(s), \ldots, c_k^{(t)}(s)\big) \quad \forall s \in V^k. \tag{2.6}$$

For two graphs G, G' the k-WL algorithm then decides "not isomorphic" if $\{\!\{c^{(t)}(s) \mid s \in V(G)^k\}\!\} \neq \{\!\{c^{(t)}(s') \mid s' \in V(G')^k\}\!\}$ for some t, and returns "maybe isomorphic" otherwise. Like 1-WL, k-WL decides "not isomorphic" only if $G \ncong G'$. The *Folklore k-WL* algorithm (k-FWL) differs in its update rule, which "swaps" the order of the aggregation steps [19]:

$$c_u^{(t)}(s) = (c^{(t-1)}_{(u,s_2,\ldots,s_k)}, c^{(t-1)}_{(s_1,u,s_3,\ldots,s_k)}, \ldots, c^{(t-1)}_{(s_1,\ldots,s_{k-1},u)}) \quad \forall u \in V, s \in V^k, \tag{2.7}$$

$$c^{(t)}(s) = \text{Hash}\big(c^{(t-1)}(s), \{\!\{c_u^{(t)}(s) \mid u \in V\}\!\}\big) \quad \forall s \in V^k. \tag{2.8}$$

The 1-WL and 2-WL test are equivalent, and for $k \geq 2$, $(k+1)$-WL can distinguish strictly more graphs than k-WL [19]. The k-FWL is as powerful as the $(k+1)$-WL for $k \geq 2$ [38].

Set-WL GNN. Since computations on k-tuples are expensive, [66] consider a GNN that corresponds to a set version of a k-WL algorithm. For any *set* $S \subseteq V$ with $|S| = k$, let $\mathcal{N}^{\text{set}}(S) = \{T \subset V, |T| = k \mid |S \cap T| = k - 1\}$. The set-based WL test (k-SWL) then updates as

$$c^{(t)}(S) = \text{Hash}\big(c^{(t-1)}(S), \{\!\{c^{(t-1)}(T) \mid T \in \mathcal{N}^{\text{set}}(S)\}\!\}\big); \tag{2.9}$$

its GNN analogue uses the aggregation and update (cf. equations (1.6) and (1.8))

$$h_S^{(t+1)} = \sigma\Big(W_1^{(t)} h_S^{(t)} + \sum\nolimits_{T \in \mathcal{N}^{\text{set}}(S)} W_2^{(t)} h_T^{(t)}\Big), \tag{2.10}$$

where σ is a coordinatewise nonlinearity (e.g., sigmoid or ReLU). This family of GNNs is equivalent in power to the k-SWL test [66] (Theorem 8). For computational efficiency, a local version restricts the neighborhood of S to sets T such that the nodes $\{u, v\} = S \Delta T$ in the symmetric difference are connected in the graph. This local version is weaker [1].

Folklore WL GNN. In analogy to the k-FWL algorithm, Maron et al. [62] define k-FGNNs with aggregations

$$h_s^{(t+1)} = f_{\mathrm{Up}}^{(t+1)}\left(h_s^{(t)}, \sum_{v \in V} \prod_{i=1}^{k} f_i^{(t+1)}\big(h^{(t)}_{(s_1,\ldots,s_{i-1},v,s_{i+1},\ldots,s_k)}\big)\right). \tag{2.11}$$

For $k = 2$, this model can be implemented via matrix multiplications. The input to the aggregation, for all pairs of nodes simultaneously, is a tensor $H \in \mathbb{R}^{n \times n \times d_t}$, with $H_{(u,v),:} = h_{(u,v)}$. The initial $H^{(0)} \in \mathbb{R}^{n \times n \times (d+1)}$ is defined as in the beginning of Section 2.3.

To compute the aggregation layer, first, we apply three MLPs $m_1, m_2 : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ and $m_3 : \mathbb{R}^{d_1} \to \mathbb{R}^{d_3}$ to each embedding $h_{(u,v)}$ in H: $m_l(H)_{(u,v),:} = m_l(H_{(u,v),:})$ for $1 \leq l \leq 3$. Then one computes an intermediate representation $H' \in \mathbb{R}^{n \times n \times d_2}$ by multiplying matching "slices" of the outputs of m_1, m_2: $H'_{:,:,i} = m_1(H)_{:,:,i} \cdot m_2(H)_{:,:,i}$. The final output of the aggregation is the concatenation $(m_3(H), H') \in \mathbb{R}^{n \times n \times (d_2+d_3)}$. A variation of this model, a low-rank global *attention* model, was shown to relate attention and the 2-FWL algorithm via *algorithmic alignment*, which we discuss in Section 3.3 [71]. Attention in neural networks introduces learned pair-wise weights in the aggregation function.

The family of k-FGNNs is a class of nonlinear equivariant networks, and is equivalent in power to the k-FWL test and the $(k+1)$-WL test [7,62] (Theorem 8).

2.3.2. Linear equivariant layers

While the models discussed so far rely on message passing, the GNN definition (2.4) only requires permutation equivariant or invariant operations in each layer. The k-linear (equivariant) GNNs (k-LEGNNs), introduced in [63], allow more general linear equivariant operations. In k-LEGNNs, each layer $F^{(t)} = \sigma \circ L^{(t)} : \mathbb{R}^{n^k \times d_{t-1}} \to \mathbb{R}^{n^k \times d_t}$ is a concatenation of a linear equivariant function $L^{(t)}$ and a coordinatewise nonlinear activation function. The function σ may also be replaced with a nonlinear function $f_1^{(t)} : \mathbb{R}^{d_{t+1/2}} \to \mathbb{R}^{d_{t+1}}$ (an MLP) applied separately to each tuple embedding $L^{(t)}(H^{(t-1)})_{s,:}$.

Characterizations of equivariant functions or networks were studied in [40,52,53,74]. Maron et al. [63] explicitly characterize all invariant and equivariant linear layers, and show that the vector space of linear invariant or equivariant functions $f : \mathbb{R}^{n^k} \to \mathbb{R}^{n^\ell}$ has dimension $\mathrm{b}(k)$ and $\mathrm{b}(k+\ell)$, respectively, where $\mathrm{b}(k)$ is the kth Bell number. When including multiple channels and bias terms, one obtains the following bounds.

Theorem 7 ([63]). *The space of invariant (equivariant) linear layers $\mathbb{R}^{n^k \times d} \to \mathbb{R}^{d'}$ ($\mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^k \times d'}$) has dimension $dd'\mathrm{b}(k) + d'$ (for equivariant, $dd'\mathrm{b}(2k) + d'\mathrm{b}(k)$).*

The GNN model uses one parameter (coefficient) for each basis tensor. Importantly, the number of parameters is independent of the number of nodes. The proof for identifying the basis tensors sets up a fixed point equation with Kronecker products of any permutation matrix that any equivariant tensor must satisfy. The solutions to these equations are defined by equivalence classes of multiindices in $[n]^k$. Each equivalence class is represented by a partition γ of $[k]$, e.g., $\gamma = \{\{1\},\{2,3\}\}$ includes all multiindices (i_1, i_2, i_3) where $i_1 \neq i_2, i_3$ and $i_2 = i_3$. The basis tensors $B^\gamma \in \{0,1\}^{n^k}$ are then such that $B_s^\gamma = 1$ if and only if $s \in \gamma$.

Linear equivariant GNNs of order k (k-LEGNNs) parameterized with the full basis are as discriminative as the k-WL algorithm [62] (Theorem 8). To achieve this discriminative power, each entry $H_{s,:}^{(0)}$ in the input tensor encodes an initial coloring of the isomorphism type of the subgraph indexed by the k-tuple s.

2.3.3. Summary of representational power via WL

The following theorem summarizes equivalence results between the GNNs discussed so far and variants of the WL test. Following [7], we here use equivalence relations, as they suffice for universal approximation in Section 2.4. For a set $\mathcal{F}$ of functions defined on $\mathcal{G}$, define an equivalence relation ρ via the joint discriminative power of all $F \in \mathcal{F}$, i.e., for any $G, G' \in \mathcal{G}$:

$$(G, G') \in \rho(\mathcal{F}) \iff \forall F \in \mathcal{F},\ F(G) = F(G'). \tag{2.12}$$

Theorem 8. *The above GNN families have the following equivalences:*

$$\rho(\mathit{MGNN}) = \rho(\mathit{2\text{-}WL}) \quad [95], \tag{2.13}$$

$$\rho(k\text{-}\mathit{set\text{-}GNN}) = \rho(k\text{-}\mathit{SWL}) \quad [66], \tag{2.14}$$

$$\rho(k\text{-}\mathit{LEGNN}) = \rho(k\text{-}\mathit{WL}) \quad [34, 63], \tag{2.15}$$

$$\rho(k\text{-}\mathit{FGNN}) = \rho((k+1)\text{-}\mathit{WL}) \quad [7, 62]. \tag{2.16}$$

Analogous results hold for equivariant models (for node representations), with the exception of equality (2.15), which becomes an inclusion: $\rho(k\text{-LEGNN}_E) \subseteq \rho(k\text{-WL}_E)$ [7].

2.3.4. Relational pooling

One option to obtain nonlinear permutation invariant functions is to average permutation-sensitive functions over the permutation group Π_n. Murphy et al. [67, 68] propose such a model, inspired by joint exchangeability of random variables [2, 29]. Concretely, if $A \in \mathbb{R}^{n \times n}$ denotes the adjacency matrix of the input graph G and $X \in \mathbb{R}^{n \times d}$ the matrix of node attributes, then

$$F_{\mathrm{RP}}(G) = \frac{1}{n!} \sum_{\pi \in \Pi_n} g(A_{\pi,\pi}, X_\pi) = g(\pi \cdot H^{(0)}), \tag{2.17}$$

where X_π is X with permuted rows, and $H^{(0)}$ is the tensor combining adjacency matrix and node attributes. Here, g is any permutation-sensitive function, and may be modeled via various nonlinear function approximators, e.g., neural networks such as fully connected networks (MLPs), recurrent neural networks or a combination of a convolutional network applied to A and an MLP applied to X. In particular, this model allows implementing graph isomorphism testing via node IDs (cf. Section 2.2) if g is a universal approximator [68]. For instance, node IDs may be permuted over nodes and concatenated with the node attributes:

$$F_{\mathrm{RP}}(G) = \frac{1}{n!} \sum_{\pi \in \Pi_n} \big(A_{\pi,\pi}, [X_\pi, I_n]\big) = \frac{1}{n!} \sum_{\pi \in \Pi_n} g\big(A, \big[X, (I_n)_\pi\big]\big), \tag{2.18}$$

where $I_n \in \mathbb{R}^{n \times n}$ is the identity matrix. If g is an MPNN, the resulting model is strictly more powerful than the 1-WL test and hence g by itself.

The drawback of the *Relational Pooling* (2.17) is its computational intractability. Various approximations have been considered, e.g., defining canonical orders, stochastic approximations, and applying g to all possible k-subsets of V. In the latter case, increasing k strictly increases the expressive power. *Local Relational Pooling* is a variant that applies relational pooling to the k-hop subgraphs centered at each node, and then aggregates the results. This operation provably allows to identify and count subgraphs of size up to k [22].

2.3.5. Recursion

A general strategy for encoding a graph is to encode a collection of subgraphs, and then aggregate these encodings. When doing so, an important bit of information are node correspondences across subgraphs [15, 86]. Otherwise, this process includes the reconstruction hypothesis [46, 88], i.e., the question whether any graph G can be reconstructed from the collection of its subgraphs $G \setminus \{v\}$, for all v in G.

Indeed, the expressive power of such a model depends on the set of subgraphs, the type of subgraph encodings and the aggregation. Tahmasebi et al. [86] show that recursion can be a powerful tool: instead of iterative message passing or layering, a *recursive* application of the above subgraph embedding step, even with a simple set aggregation like (1.6), can enable a GNN that can count any bounded-size subgraphs, as opposed to MPNNs (Proposition 2).

Let $\mathcal{N}_r(v)$ be the r-hop neighborhood of v in G. *Recursive neighborhood pooling (RNP)* encodes *intersections* of such neighborhoods of different radii. Given an input graph G with node attributes $\{h_u^{\text{in}}\}_{u \in V(G)}$ and a sequence $(r_1, \ldots, r_t)$ of radii, RNP recursively encodes the node-deleted r_1-neighborhoods $G_v = \mathcal{N}_{r_1}(v) \setminus \{v\}$ of all nodes $v \in V$ after marking the deletion in augmented representations h_u^{aug}, $u \in V$. It then combines the results, and returns node representations of all nodes. I.e., for each node $v \in V$, it computes G_v and

$$\begin{aligned}
&& h_u^{\text{aug}} &= \big(h_u^{\text{in}}, \mathbf{1}\big[(u,v) \in E(G_v)\big]\big) \quad \forall u \in V(G_v), && (2.19)\\
&& \{\!\{h'_{v,u}\}\!\}_{u \in G_v} &\leftarrow \text{RNP-GNN}\big(G_v, \{\!\{h_u^{\text{aug}}\}\!\}_{u \in G_v}, (r_2, r_3, \ldots, r_t)\big), \quad \text{(recursion)} && (2.20)\\
&\text{return} & h_v^{\text{out}} &= f_{\text{Agg}}^{(t)}\big(h_v^{\text{in}}, \{\!\{h'_{v,u}\}\!\}_{u \in G_v}\big), \quad \forall v \in V. && (2.21)
\end{aligned}$$

If the sequence of radii is empty (base case), then the algorithm returns the input attributes h_u^{in}. In contrast to *iterative* message passing, the encoded subgraphs here correspond to intersections of local neighborhoods. Together with the node deletions and markings that retain node correspondences, this maintains more structural information. If the radii sequence dominates a covering sequence for a subgraph H of interest, then, with appropriate parameters, RNP can count the induced and noninduced subgraphs of G isomorphic to H [86]. The computational cost is $O(n^k)$ for recursion depth k, and better for very sparse graphs, in line with computational lower bounds.

2.4. Universal approximation

Distinguishing given graphs is closely tied to approximating continuous functions on graphs. In early work, Scarselli et al. [82] take a fixed point view and show a universal approximation result for infinite-depth MPNNs whose layers are contraction operators, for functions on equivalence classes defined by computation trees. Dehmamy et al. [28] analyze the ability of GNNs to compute polynomial functions of the adjacency matrix.

Later works derive universal approximation results for graph and permutation-equivariant functions from graph discrimination results via extensions of the Stone–Weierstrass theorem [7, 23, 47, 64]. For instance, H-invariant networks (for a permutation group H) can universally approximate H-invariant polynomials [64], which in turn can universally approximate any invariant function [98]. Keriven and Peyré [47] do not fix the size of the graph and show that shallow equivariant networks can, with a single set of parameters, well approximate a function on graphs of varying size. Both constructions involve very large tensors.

More generally, the Stone–Weierstrass theorem (for symmetries) allows translating Theorem 8 into universal approximation results. Let $\mathcal{C}_I(\mathcal{X}, \mathcal{Y})$ be the set of invariant continuous functions from $\mathcal{X}$ to $\mathcal{Y}$. Then a class $\mathcal{F}$ of GNNs is *universal* if its closure $\overline{\mathcal{F}}$ (in uniform norm) on a compact set K is the entire $\mathcal{C}_I(K, \mathbb{R}^p)$.

Theorem 9 ([7]). *Let $K_{disc} \subseteq \mathcal{G}_n \times \mathbb{R}^{d_0 \times n}$, $K \subseteq \mathbb{R}^{d_0 \times n}$ be compact sets, where $\mathcal{G}_n$ is the set of all unweighted graphs on n nodes. Then*

$$\overline{MGNN} = \{f \in \mathcal{C}_I(K_{\text{disc}}, \mathbb{R}^p) : \rho(2\text{-}WL) \subseteq \rho(f)\}, \tag{2.22}$$

$$\overline{k\text{-}LEGNN} = \{f \in \mathcal{C}_I(K, \mathbb{R}^p) : \rho(k\text{-}WL) \subseteq \rho(f)\}, \tag{2.23}$$

$$\overline{k\text{-}FGNN} = \{f \in \mathcal{C}_I(K, \mathbb{R}^p) : \rho((k+1)\text{-}WL) \subseteq \rho(f)\}. \tag{2.24}$$

Analogous relations hold for equivariant functions, except for

$$\overline{k\text{-}LEGNN_E} = \{f \in \mathcal{C}_E(K, \mathbb{R}^{n \times p}) : \rho(k\text{-}LEGNN_E) \subseteq \rho(f)\},$$

which is a superset of $\{f \in \mathcal{C}_E(K, \mathbb{R}^{n \times p}) : \rho(k\text{-}WL_E) \subseteq \rho(f)\}$.

3. GENERALIZATION

Beyond approximation power, a second important question in machine learning is generalization. *Generalization* asks how well the estimated function $\hat{F}$ is performing according to the population risk, i.e., $\mathcal{R}(\hat{F})$, as a function of the number of data points N and model properties. Good generalization may demand explicit (e.g., via a penalty term) or implicit regularization (e.g., via the optimization algorithm). Hence, generalization analyses involve aspects of the complexity of the model class $\mathcal{F}$, the target function we aim to learn, the data and the optimization procedure. This is particularly challenging for neural networks, due to the nested functional form and the nonconvexity of the empirical risk.

A classic learning theoretic perspective bounds the *generalization gap* $\mathcal{R}(\hat{F}) - \widehat{\mathcal{R}}(\hat{F})$ via the complexity of the model class $\mathcal{F}$ (Section 3.1). These approaches do not take into account possible implicit regularization via the optimization procedure. One possibility to do so is via the *Neural Tangent Kernel* approximation (Section 3.2). Finally, for more complex, structured target functions, e.g., algorithms or physics simulations, one may want to also consider the structure of the target task. One such option is *Algorithmic Alignment*

(Section 3.3). Another strategy for obtaining generalization bounds is via *algorithmic stability*, the condition that, if one data point is replaced, the outcome of the learning algorithm does not change much. This strategy led to some early bounds for spectral GNNs [91].

3.1. Generalization bounds via complexity of the model class

Vapnik–Chervonenkis dimension. The first GNN generalization bound was based on bounding the Vapnik–Chervonenkis (VC) dimension [89] of the GNN function class $\mathcal{F}$. The *VC dimension* of $\mathcal{F}$ expresses the maximum size of a set of data points such that for any binary labeling of the data, some GNN in $\mathcal{F}$ can perfectly fit, i.e., *shatter*, the set. The VC dimension directly leads to a bound on the generalization gap. Here, we only state the results for sigmoid activation functions.

Theorem 10 ([84]). *The VC dimension of GNNs with p parameters, H hidden neurons (in the MLP) and input graphs of size n is $O(p^2 H^2 n^2)$.*

Strictly speaking, Theorem 10 is for node classification with one hidden layer in the aggregation function MLPs. The VC dimension directly yields a bound on the generalization gap: for a class $\mathcal{F}$ with VC dimension D, with probability $1-\delta$, it holds that $\mathcal{R}(\hat{f}) - \widehat{\mathcal{R}}(\hat{f}) \leq O(\sqrt{\frac{D}{N}\log\frac{N}{D}}) + \sqrt{\frac{1}{2N}\log\frac{1}{\delta}}$. Interestingly, in these bounds, GNNs are a generalization of recurrent neural networks [84]. The VC dimension bounds for GNNs are the same as for recurrent neural networks [50]; for fully connected MLPs, they are missing the factor n^2 [45].

Rademacher complexity. Bounds that are in many cases tighter can be obtained via Rademacher complexity. The *empirical Rademacher complexity* $\widehat{\mathfrak{R}}_S(\mathcal{F})$ of a function class $\mathcal{F}$ measures how well it can fit "noise" in the form of uniform random variables $\sigma = (\sigma_1, \dots, \sigma_N)$ in $\{-1, +1\}$: $\widehat{\mathfrak{R}}_S(\mathcal{F}) = \mathbb{E}_\sigma[\sup_{F \in \mathcal{F}} \frac{1}{N}\sum_{i=1}^{N} \sigma_i F(x_i)]$, for a fixed data sample $S = \{x_1, \dots, x_N\}$. Similarly to VC dimension, $\widehat{\mathfrak{R}}_S(\mathcal{F})$ provides a bound on the probability of error under the full data distribution: $\mathbb{P}[\mathrm{error}(F)] \leq \widehat{\mathcal{R}}(F) + 2\widehat{\mathfrak{R}}_S(\mathcal{J}) + 3\sqrt{\frac{\log(2/\delta)}{2N}}$, where $\mathcal{J}$ is the class of functions $F \in \mathcal{F}$ concatenated with the loss. Garg et al. [33] analyze a GNN that applies a logistic linear binary classifier at each node, averages these predictions for a graph-level prediction, and uses a *mean field update* [26]: $h_v^t = \phi(W_1 x_v + W_2 \rho(\sum_{u \in N(v)} g(h_u^{t-1})))$, where ϕ, ρ, g are nonlinear functions with bounded Lipschitz constant that are zero at zero (e.g., tanh), and $\|W_1\|_F, \|W_2\|_F \leq B$. The logistic predictor outputs a "probability" for the label 1, and is evaluated by a margin loss function that gives a (scaled) penalty if the "probability" of the correct label is below a threshold ($\frac{\gamma+1}{2}$).

Theorem 11 ([33]). *Let $\mathcal{C}$ be the product of the Lipschitz constants of ϕ, ρ, g, and B; T the number of GNN iterations; w the dimension of the embeddings h_v^t, and d the maximum branching factor in the computation tree. Then the generalization gap of the GNN can be bounded as: $\tilde{O}(\frac{wd}{\sqrt{N}\gamma})$ for $\mathcal{C} < 1/d$, $\tilde{O}(\frac{wdT}{\sqrt{N}\gamma})$ for $\mathcal{C} = 1/d$, and $\tilde{O}(\frac{wd\sqrt{wT}}{\sqrt{N}\gamma})$ for $\mathcal{C} > 1/d$.*

The factor d is equal to $\max_{v \in G} \deg(v) - 1$. For recurrent neural networks, the same bounds hold, but with $d = 1$ [21]: a sequence is a tree with branching factor 1. In comparison, for the VC bounds in this setting, with $H = w$, $n > d$ and p is the size of the matrices W (about w^2), we obtain a generalization bound of $\tilde{O}(w^3 n/\sqrt{N})$, ignoring log factors. Later work tightens the bounds in Theorem 11 by using a *PAC-Bayesian* approach [56].

3.2. Generalization bounds via the Neural Tangent Kernel

Infinitely-wide neural networks can be related to kernel learning techniques [4,5,31, 32,43]. Du et al. [30] extend this analysis to a broad class of GNNs. The main idea underlying the *Neural Tangent Kernel (NTK)* is to approximate a neural network $F(\theta, G)$ with a kernel derived from the training dynamics. Assume we fit $F(\theta, G)$ with the squared loss $L(\theta) = \sum_{i=1}^N \ell(F(\theta, G_i), y_i) = \frac{1}{2}(F(\theta, G_i) - y_i)^2$, where $\theta \in \mathbb{R}^m$ collects all parameters of the network. If we optimize with gradient descent with infinitesimally small step size, i.e., $\frac{d\theta(t)}{dt} = -\nabla L(\theta(t))$, then the network outputs $u(t) = (F(\theta(t), G_i))_{i=1}^N$ follow the dynamics

$$\frac{du}{dt} = -H(t)\big(u(t) - \mathbf{y}\big), \quad \text{where } H(t)_{ij} = \left\langle \frac{\partial F(\theta(t), G_i)}{\partial \theta}, \frac{\partial F(\theta(t), G_j)}{\partial \theta} \right\rangle. \tag{3.1}$$

Here, $\mathbf{y} = (y_i)_{i=1}^N$. If θ is sufficiently large (i.e., the network sufficiently wide), then it was shown that the matrix $H(t) \in \mathbb{R}^{N \times N}$ remains approximately constant as a function of t. In this case, the neural network becomes approximately a kernel regression [85]. If the parameters $\theta(0)$ are initialized as i.i.d. Gaussian, then the matrix $H(0)$ converges to a deterministic kernel matrix $\tilde{H}$, the *Neural Tangent Kernel*, with closed form regression solution $F_{\tilde{H}}(G)$. Given this approximation, one may analyze generalization via kernel learning theory.

Theorem 12 ([11]). *Given N i.i.d. training data points and any loss function $\ell : \mathbb{R} \times \mathbb{R} \to [0, 1]$ that is 1-Lipschitz in the first argument with $\ell(y, y) = 0$, with probability $1 - \delta$ the population risk of the Graph Neural Tangent predictor is bounded as*

$$\mathcal{R}(F_{\tilde{H}}) = O\left(\frac{1}{N}\sqrt{\mathbf{y}^\top \tilde{H}^{-1}\mathbf{y} \cdot \operatorname{tr}(\tilde{H})} + \sqrt{\frac{1}{N}\log(1/\delta)}\right).$$

In contrast to the results in Section 3.1, the complexity measure $\mathbf{y}^\top \tilde{H}^{-1}\mathbf{y}$ of the target function is data-dependent. If the target function to be learned follows a simple GNN structure with a polynomial, then this bound can be polynomial:

Theorem 13 ([30]). *Let $\bar{h}_v = c_v \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u$. If the labels y_i, $1 \leq i \leq N$, satisfy*

$$y_i = \alpha_1 \sum_{v \in V(G_i)} \beta_1^\top \bar{h}_v + \sum_{l=1}^{\infty} \alpha_{2l} \sum_{v \in V} \left(\beta_{2l}^\top \bar{h}_v\right)^{2l}$$

for $\alpha_k \in \mathbb{R}$, $\beta_k \in \mathbb{R}^d$, then $\mathbf{y}^\top \tilde{H}^{-1}\mathbf{y} \leq 2|\alpha_1| \cdot \|\beta_1\|_2 + \sum_{l=1}^{\infty} \sqrt{2\pi}(2l - 1)|\alpha_{2l}| \cdot \|\beta_{2l}\|_2^{2l}$. With $n = \max_i V(G_i)$, we have $\operatorname{tr}(\tilde{H}) = O(n^2 N)$.

3.3. Generalization via algorithmic alignment

The Graph NTK analysis shows a polynomial sample complexity if the function to be learned is close to the computational structure of the GNN, in a simple way. While this applies to mainly simpler learning tasks, the idea of an "alignment" of computational structure carries further. Recently, there has been growing interest in learning scientific tasks, e.g., given a set of particles or planets along with their location, mass and velocity, predict the next state of the system [12,78,79], and in "algorithmic reasoning," e.g., learning to solve combinatorial optimization problems in particular over graphs [20]. In such cases, the target function corresponds to an algorithm, e.g., dynamic programming.

While many neural network architectures have the power to represent such tasks, empirically, they do not learn them equally well from data. In particular, GNNs perform well here, i.e., their architecture encodes suitable *inductive biases* [13,96]. As a concrete example, consider the Shortest Path problem. The computational structure of MPNNs matches that of the Bellman–Ford (BF) algorithm [14] very well: both "algorithms" iterate, and in each iteration t, update the state as a function of the neighboring nodes and edge weights $w(u, v)$:

$$\begin{aligned} &\text{(BF)} \;\; d[t][v] = \min_{u \in \mathcal{N}(v)} d[t-1][u] + w(u,v), \\ &\text{(GNN)} \;\; h_v^t = \sum_{u \in \mathcal{N}(v)} \mathrm{MLP}\big(h_u^{t-1}, h_v^{t-1}, w(u,v)\big). \end{aligned} \tag{3.2}$$

Hence, the GNN can simulate the BF algorithm if it uses sufficiently many iterations, and if the aggregation function approximates the BF state update. Intuitively, this is a much simpler function to learn than the full algorithm as a black box, i.e., the GNN encodes much of the algorithmic structure, sparsity and invariances in the architecture. More generally, MPNNs match the structure of many dynamic programs in an analogous way [96].

The NTK results formalize simplicity by a small function norm in the RKHS associated with the Graph NTK; this can become complicated with more complex tasks and multiple layers. To quantify *structural* match, Xu et al. [96] define *algorithmic alignment* by viewing a neural network as a structured arrangement of learnable modules – in a GNN, the (MLPs in the) aggregation functions – and define complexity via sample complexity of those modules in a PAC-learning framework. Sample complexity in PAC learning is defined as follows: We are given a data sample $\{(x_i, y_i)\}_{i=1}^N$ drawn i.i.d. from a distribution $\mathcal{P}$ that satisfies $y_i = g(x_i)$ for an underlying target function g. Let $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^N)$ be the function output by a learning algorithm $\mathcal{A}$. For a fixed error ε and failure probability $1 - \delta$, the function g is (N, ε, δ)-*PAC learnable* with $\mathcal{A}$ if

$$\mathbb{P}_{x \sim \mathcal{P}}\big[\big|f(x) - g(x)\big| < \varepsilon\big] \geq 1 - \delta. \tag{3.3}$$

The *sample complexity* $\mathcal{C}_{\mathcal{A}}(g, \varepsilon, \delta)$ is the smallest N so that g is (N, ε, δ)-learnable with $\mathcal{A}$.

Definition 1 (Algorithmic alignment [96]). Let g be a target function and $\mathcal{N}$ a neural network with M modules $\mathcal{N}_i$. The module functions $f_1, \dots, f_M$ generate g for $\mathcal{N}$ if, by replacing $\mathcal{N}_i$ with f_i, the network $\mathcal{N}$ simulates g. Then $\mathcal{N}$ (N, ε, δ)-*algorithmically aligns* with g if (1) $f_1, \dots, f_M$ generate g and (2) there are learning algorithms $\mathcal{A}_i$ for the $\mathcal{N}_i$'s such that $M \cdot \max_i C_{\mathcal{A}_i}(f_i, \varepsilon, \delta) \leq N$.

Algorithmic alignment resembles Kolmogorov complexity [51]. Thus, it can be hard to obtain the optimal alignment between a neural network and an algorithm. But, *any* algorithmic alignment yields a bound, and any with acceptable sample complexity may suffice.

The complexity of the MLP modules in GNNs may be measured with a variety of techniques. One option is the NTK framework. The module-based bounds then resemble the polynomial bound in Theorem 13, since both are extensions of [5]. However, here, the bounds are applied at a module level, and not for the entire GNN as a unit. Theorem 14 translates these bounds, in a simplified setting, into sample complexity bounds for the full network.

Theorem 14 ([96]). *Fix ε and δ. Suppose $\{(G_i, y_i)\}_{i=1}^N \sim \mathcal{P}$, where $|V(G_i)| < n$, and $y_i = g(G_i)$ for some g. Suppose $\mathcal{N}_1, \dots, \mathcal{N}_M$ are network $\mathcal{N}$'s MLP modules in sequential order of processing. Suppose $\mathcal{N}$ and g (N, ε, δ)-algorithmically align via functions $f_1, \dots, f_M$ for a constant M. Under the following assumptions, g is $(N, O(\varepsilon), O(\delta))$-learnable by $\mathcal{N}$.*

(a) Sequential learning. *We train $\mathcal{N}_i$'s sequentially: $\mathcal{N}_1$ has input samples $\{\hat{x}_i^{(1)}, f_1(\hat{x}_i^{(1)})\}_{i=1}^N$, with $\hat{x}_i^{(1)}$ obtained from G_i. For $j > 1$, the input $\hat{x}_i^{(j)}$ for $\mathcal{N}_j$ are the outputs of the previous modules, but labels are generated by the correct functions $f_{j-1}, \dots, f_1$ on $\hat{x}_i^{(1)}$.*

(b) Algorithm stability. *Let $\mathcal{A}$ be the learning algorithm for the $\mathcal{N}_i$'s, $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^N)$, and $\hat{f} = \mathcal{A}(\{\hat{x}_i, y_i\}_{i=1}^N)$. For any x, $\|f(x) - \hat{f}(x)\| \le L_0 \cdot \max_i \|x_i - \hat{x}_i\|$, for some $L_0 < \infty$.*

(c) Lipschitzness. *The learned functions $\hat{f}_j$ satisfy $\|\hat{f}_j(x) - \hat{f}_j(\hat{x})\| \le L_1 \|x - \hat{x}\|$, for some $L_1 < \infty$.*

The big O notation here hides factors including the Lipschitz constants, number of modules, and graph size. When measuring module complexity via the NTK, Theorem 14, e.g., indeed yields a gap between fully connected networks and GNNs in simple cases [96], supporting empirical results. While some works use sequential training [90], empirically, better alignment improves learning and generalization in practice even with more common "end-to-end" training, i.e., optimizing all parameters simultaneously [13,96].

At a general level, these alignment results indicate that it is not only possible to learn combinatorial algorithms and physical reasoning tasks with machine learning, but how, in turn, incorporating expert knowledge, e.g., in algorithmic techniques or physics, into the design of the learning method can improve sample efficiency.

4. EXTRAPOLATION

Section 3 summarizes results for in-distribution generalization, i.e., how well a learned model performs on data from the same distribution $\mathcal{P}$ as the training data. Yet, in many practical scenarios, a model is applied to data from a different distribution. A strong case of such a distribution shift is *extrapolation*. It considers the expected loss $\mathbb{E}_{(G,y)\sim\mathcal{Q}}[\ell(G, y, F(G))]$ under a distribution $\mathcal{Q}$ with different support, e.g., $\mathrm{supp}(\mathcal{Q}) \supset \mathrm{supp}(\mathcal{P})$. For graphs, $\mathcal{Q}$ may entail graphs of different sizes, different degrees, or with node attributes in different ranges from the training graphs. As no data has been observed in the new domain parts, extrapolation can be ill-defined without stronger assumptions on the task and model class. What assumptions are sufficient? Theoretical results on extrapolation assume the graphs have sufficient structural similarity and/or the model class is sufficiently restricted to extrapolate accurately. Empirically, while extrapolation has been difficult, several works achieve GNN extrapolation in tasks like predicting the time evolution of physical systems [12], learning graph algorithms [90], and solving equations [54].

Structural similarity of graphs. One possibility to guarantee successful extrapolation to larger graphs is to assume sufficient structural similarity between the graphs in $\mathcal{P}$ and $\mathcal{Q}$, in particular, structural properties that matter for the GNN family under consideration. For spectral GNNs, which learn functions of the graph Laplacian, this assumption has been formalized as the graphs arising from the same underlying topological space, manifold or graphon. Under such conditions, spectral GNNs – with conditions on the employed filters – can generalize to larger graphs [55,76,77].

For message passing GNNs, whose representations rely on computation trees as local structures (Section 2.1), an agreement in the distributions of the computation trees in the graphs sampled from $\mathcal{P}$ and $\mathcal{Q}$ is necessary [99]. This is violated, for instance, if the degree distribution is a function of the graph size, as is the case for random graphs under the Erdős–Rényi or Preferential Attachment models. The computation tree of depth t rooted at a node v corresponds to the color $c^{(t)}(v)$ assigned by the 1-WL algorithm.

Theorem 15 ([99]). *Let $\mathcal{P}$ and $\mathcal{Q}$ be finitely supported distributions of graphs. Let $\mathcal{P}^t$ be the distribution of colors $c^{(t)}(v)$ over $\mathcal{P}$ and similarly $\mathcal{Q}^t$ for $\mathcal{Q}$. Assume that any graph in $\mathcal{Q}$ contains a node with a color in $\mathcal{Q}^t \setminus \mathcal{P}^t$. Then, for any graph regression task solvable by a GNN with depth t there exists a GNN with depth at most $t+3$ that perfectly solves the task on $\mathcal{P}$ and predicts an answer with arbitrarily large error on all graphs from $\mathcal{Q}$.*

The proof exploits the fact that GNN predictions on nodes only depend on the associated computation tree, and that a sufficiently flexible GNN (depth at least $t+2$ layers and width $\max\{(\max \deg(G)+1)^t \cdot |C|, 2\sqrt{|P|}\}$, where the max degree refers to any graph in the support, $|C|$ is the finite number of possible input node attributes and P the set of colors encountered in graphs in the support) can assign arbitrary target labels to any computation tree [66,99]. That is, the available information allows for multiple local minima of the empirical risk. A similar result can be shown for node prediction tasks.

Conditions on the GNN. If one cannot guarantee sufficient structural similarity of the input graphs, then further restrictions on the GNN model can enable extrapolation to different graph sizes, structures and ranges of input node attributes. If there are no training observations in a certain range of attributes or local structures, then the predictions of the learned model depend on the *inductive biases* induced by the model architecture, loss function and training algorithm. In other words, which, out of multiple fitting functions (minima), a model will choose, depends on these biases.

Xu et al. [97] analyze such biases to obtain conditions on the GNN for extrapolation. Taking the perspective of algorithmic alignment (Section 3.3), they first analyze how individual module functions, i.e., the MLPs in the aggregation function of a GNN, extrapolate, and then transfer this to the entire GNN. The aggregation functions enter the extrapolation regime, e.g., if the node attributes, node degrees or computation trees are different in $\mathcal{Q}$, as they determine the inputs to the aggregations. The following theorem states that, away from $\mathrm{supp}(\mathcal{P})$, MLPs implement directionally linear functions.

Theorem 16 ([97]). *Suppose we train a two-layer MLP $f : \mathbb{R}^d \to \mathbb{R}$ with ReLU activation functions with squared loss in the NTK regime. For any direction $v \in \mathbb{R}^d$, let $x_0 = tv$. As $t \to \infty$, $f(x_0 + hv) - f(x_0) \to \beta_v \cdot h$ for any $h > 0$, where β_v is a constant linear coefficient. Moreover, given $\varepsilon > 0$, for $t = O(\frac{1}{\varepsilon})$, we have $|\frac{f(x_0+hv)-f(x_0)}{h} - \beta_v| < \varepsilon$.*

The linear function and the constant terms in the convergence rate depend on the training data and the direction v. The proof of Theorem 16 relies on the fact that a neural network in the NTK regime learns a minimum-norm interpolation function [4,5,43]. Although Theorem 16 uses a simplified setting of a wide 2-layer network, similar results hold empirically for more general MLPs [97].

To appreciate the implications of this result in the context of GNNs, consider the example of Shortest Path in equation (3.2). For the aggregation function to mimic the Bellman–Ford algorithm, the MLP must approximate a nonlinear function. But, in the extrapolation regime, it implements a linear function and therefore is expected to not approximate Bellman–Ford well any more. Indeed, empirical works that successfully extrapolate GNNs for Shortest Path use a different aggregation function of the form [13,90]

$$h_u^{(t)} = \min_{v \in \mathcal{N}(u)} \mathrm{MLP}^{(t)}\big(h_u^{(t-1)}, h_v^{(t-1)}, w_{(v,u)}\big). \tag{4.1}$$

Here, the nonlinear parts do not need to be learned, allowing to extrapolate with a linear learned MLP. More generally, the directionally linear extrapolation suggests that the (1) architecture or (2) input encoding should be set up such that the target function can be approximated by MLPs learning linear functions (*linear algorithmic alignment*). An example for (2) may be found in forecasting physical systems, e.g., predicting the evolution of n objects in a gravitational system, and the node (object) attributes are mass, location, and velocity at time t. The position of an object at time $t + 1$ is a nonlinear function of the attributes of the other objects. When encoding the nonlinear function as transformed edge attributes, the function to be learned becomes linear. Indeed, many empirical works that successfully extrapolate implement the idea of linear algorithmic alignment [24,44,61,87,97,100].

Finally, the geometry of the training data also plays an important role. Xu et al. [97] show empirical results and initial theoretical results for learning max-degree, suggesting that, even with linear algorithmic alignment, sufficient diversity in the training data is needed to identify the correct linear functions.

For the case when the target test distribution $\mathcal{Q}$ is known, Yehudai et al. [99] propose approaches for combining elements of $\mathcal{P}$ and $\mathcal{Q}$ to enhance the range of the data seen by the GNN.

5. CONCLUSION

This survey covered three main topics in understanding GNNs: representation, generalization, and extrapolation. As GNNs are an active research area, many results could not be covered. For example, we focused on MPNNs and main ideas for higher-order GNNs, but neglected spectral GNNs, which closely relate to ideas in graph signal processing. Other emergent topics include adversarial robustness, optimization behavior of the empirical risk and its improvements, and computational scalability and approximations. Moreover, GNNs have a rich set of mathematical connections, a selection of which was summarized here.

For function approximation, the limitations of MPNNs motivated powerful higher-order GNNs. However, these are still computationally expensive. What efficiency is theoretically possible? Moreover, most applications may not require full graph isomorphism power, or k-WL power for large k. What other measures make sense? Do they allow better and sharper complexity results? Initial works consider, e.g., subgraph counting [22, 86].

The generalization results so far need to use simplifications in the analysis. To what extent can they be relaxed? Do more specific tasks or graph classes allow sharper results? Which modifications of GNNs would allow them to generalize better, and how do higher-order GNNs generalize? Similar questions pertain to extrapolation and reliability under distribution shifts, a topic that has been studied even less than GNN generalization.

In general, revealing further mathematical connections may enable the design of richer models and enable a more thorough understanding of GNNs' learning abilities and limitations, and potential improvements.

ACKNOWLEDGMENTS

The author would like to thank Keyulu Xu, Derek Lim, Behrooz Tahmasebi, Vikas Garg, Tommi Jaakkola, Andreas Loukas, Joan Bruna, and Yusu Wang for many discussions on the theory of GNNs, and Jingling Li, Mozhi Zhang, Simon Du, Ken-ichi Kawarabayashi, Weihua Hu, and Jure Leskovec for collaborations.

FUNDING

This work was partially supported by NSF CAREER award 1553284, SCALE MoDL award 2134108, and CCF-2112665 (TILOS AI Research Institute).

REFERENCES

[1] R. Abboud, I. I. Ceylan, M. Grohe, and T. Lukasiewicz, The surprising power of graph neural networks with random node initialization. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[2] D. J. Aldous, Representations for partially exchangeable arrays of random variables. *J. Multivariate Anal.* **11** (1981), no. 4, 581–598.

[3] D. Anglouin, Local and global properties in networks of processors. In *Symposium on Theory of Computing (STOC)*, 1980.

[4] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang, On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[5] S. Arora, S. S. Du, W. Hu, Z. Li, and R. Wang, Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *Int. Conference on Machine Learning (ICML)*, 2019.

[6] V. Arvind, J. Köbler, G. Rattan, and O. Verbitsky, On the power of color refinement. In *International Symposium on Fundamentals of Computation Theory (FCT)*, pp. 339–350, Springer International Publishing, 2015.

[7] W. Azizian and M. Lelarge, Expressive power of invariant and equivariant graph neural networks. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

[8] L. Babai, P. Erdős, and S. M. Selkow, Random graph isomorphism. *SIAM J. Comput.* **9** (1980), no. 3, 628–635.

[9] L. Babai and L. Kučera, Canonical labelling of graphs in linear average time. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1979.

[10] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. L. Reutter, and J. P. Silva, The logical expressiveness of graph neural networks. In *Int. Conf. on Learning Representations (ICLR)*, 2020.

[11] P. L. Bartlett and S. Mendelson, Rademacher and Gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.* **3** (2002), 463–482.

[12] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu, Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4502–4510, 2016.

[13] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, Relational inductive biases, deep learning, and graph networks. 2018, arXiv:1806.01261v3.

[14] R. Bellman, On a routing problem. *Quart. Appl. Math.* **16** (1958), 87–90.

[15] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron, Equivariant Subgraph Aggregation Networks. 2021, arXiv:2110.02910v2.

[16] C. Bodnar, F. Frasca, N. Otter, Y. Guang Wang, P. Liò, G. Montúfar, and M. Bronstein, Weisfeiler and Lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[17] C. Bodnar, F. Frasca, N. Otter, Y. Guang Wang, P. Liò, G. Montúfar, and M. Bronstein, Weisfeiler and Lehman go topological: Message passing simplicial networks. In *Int. Conference on Machine Learning (ICML)*, 2021.

[18] J.-Y. Cai, M. Fürer, and N. Immerman, An optimal lower bound on the number of variables for graph identification. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989.

[19] J.-Y. Cai, M. Fürer, and N. Immerman, An optimal lower bound on the number of variables for graph identification. *Combinatorica* **12** (1992), no. 4, 389–410.

[20] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković, Combinatorial optimization and reasoning with graph neural networks. 2021, arXiv:2102.09544.

[21] M. Chen, X. Li, and T. Zhao, On generalization bounds of a family of recurrent neural networks. In *Proc. Int. Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[22] Z. Chen, L. Chen, S. Villar, and J. Bruna, Can graph neural networks count substructures? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[23] Z. Chen, S. Villar, L. Chen, and J. Bruna, On the equivalence between graph isomorphism testing and function approximation with GNNs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[24] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, Discovering symbolic models from deep learning with inductive biases. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[25] G. Cybenko, Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems* **2** (1989), no. 4, 303–314.

[26] H. Dai, B. Dai, and L. Song, Discriminative embeddings of latent variable models for structured data. In *Int. Conference on Machine Learning (ICML)*, 2016.

[27] G. Dasoulas, L. Dos Santos, K. Scaman, and A. Virmaux, Coloring graph neural networks for node disambiguation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

[28] N. Dehmamy, A. L. Barabási, and R. Yu, Understanding the representation power of graph neural networks in learning graph topology. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[29] P. Diaconis and S. Janson, Graph limits and exchangeable random graphs. *Rend. Mat. Appl.* **VII** (2008), 33–61.

[30] S. S. Du, K. Hou, R. R. Salakhutdinov, B. Poczos, R. Wang, and K. Xu, Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[31] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, Gradient descent finds global minima of deep neural networks. In *Int. Conference on Machine Learning (ICML)*, 2019.

[32] S. S. Du, X. Zhai, B. Poczos, and A. Singh, Gradient descent provably optimizes over-parameterized neural networks. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

[33] V. K. Garg, S. Jegelka, and T. Jaakkola, Generalization and representational limits of graph neural networks. In *Int. Conference on Machine Learning (ICML)*, 2020.

[34] F. Geerts, The expressive power of kth-order invariant graph networks. 2020, arXiv:2007.12035.

[35] F. Geerts, F. Mazowiecki, and G. A. Pérez, Let's agree to degree: Comparing graph convolutional networks in the message-passing framework. In *Int. Conference on Machine Learning (ICML)*, 2021.

[36] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, Neural message passing for quantum chemistry. In *Int. Conference on Machine Learning (ICML)*, 2017.

[37] M. Gori, G. Monfardini, and F. Scarselli, A new model for learning in graph domains. In *International Joint Conference on Neural Networks (IJCNN)*, 2005.

[38] M. Grohe and M. Otto, Pebble games and linear equations. *J. Symbolic Logic* **80** (2015), no. 3, 797–844.

[39] W. L. Hamilton, R. Ying, and J. Leskovec, Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[40] J. S. Hartford, D. R. Graham, K. Leyton-Brown, and S. Ravanbakhsh, Deep models of interactions across sets. In *Int. Conference on Machine Learning (ICML)*, 2018.

[41] L. Hella, M. Järvisalo, A. Kuusisto, J. Laurinharju, T. Lempiắinen, K. Luosto, J. Suomela, and J. Virtema, Weak models of distributed computing, with connections to modal logic. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 185–194, Association for Computing Machinery New York, NY, United States, 2012.

[42] N. Immerman and E. S. Lander, Describing graphs: a first-order approach to graph canonization. In *Complexity theory retrospective*, pp. 59–81, Springer, 1990.

[43] A. Jacot, F. Gabriel, and C. Hongler, Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[44] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, F. Li, C. L. Zitnick, and R. Girshick, Inferring and executing programs for visual reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[45] M. Karpinski and A. Macintyre, Polynomial bounds for the VC dimension of sigmoidal and general Pfaffian networks. *J. Comput. System Sci.* **54** (1997), no. 1, 169–176.

[46] P. Kelly, A congruence theorem for trees. *Pacific J. Math.* **7** (1957), 961–968.

[47] N. Keriven and G. Peyré, Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[48] S. Kiefer, P. Schweitzer, and E. Selman, Graphs identified by logics with counting. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2015.

[49] T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks. In *Int. Conf. on Learning Representations (ICLR)*, 2017.

[50] P. Koiran and E. D. Sontag, Vapnik–Chervonenkis dimension of recurrent neural networks. In *European Conference on Computational Learning Theory*, pp. 223–237, 1997.

[51] A. N. Kolmogorov, On tables of random numbers. *Theoret. Comput. Sci.* **207** (1998), no. 2, 387–395.

[52] R. Kondor, H. Truong Son, H. Pan, B. M. Anderson, and S. Trivedi, Covariant compositional networks for learning graphs. In *International Conference on Learning Representations (ICLR) – Workshop Track*, 2018.

[53] R. Kondor and S. Trivedi, On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *Int. Conference on Machine Learning (ICML)*, 2018.

[54] G. Lample and F. Charton, Deep learning for symbolic mathematics. In *Int. Conf. on Learning Representations (ICLR)*, 2020.

[55] R. Levie, W. Huang, L. Bucci, M. M. Bronstein, and G. Kutyniok, Transferability of spectral graph convolutional neural networks. *J. Mach. Learn. Res.* (2021).

[56] R. Liao, R. Urtasun, and R. Zemel, A PAC-Bayesian approach to generalization bounds for graph neural networks. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

[57] N. Linial, Locality in distributed graph algorithms. *SIAM J. Comput.* **21** (1992), no. 1, 193–201.

[58] A. Loukas, How hard is to distinguish graphs with graph neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[59] A. Loukas, What graph neural networks cannot learn: depth vs width. In *Int. Conf. on Learning Representations (ICLR)*, 2020.

[60] A. Magner, M. Baranwal, and A. O. Hero, The power of graph convolutional networks to distinguish random graph models. In *IEEE International Symposium on Information Theory (ISIT)*, 2020.

[61] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu, The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

[62] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, Provably powerful graph networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[63] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, Invariant and equivariant graph networks. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

[64] H. Maron, E. Fetaya, N. Segol, and Y. Lipman, On the universality of invariant networks. In *Int. Conference on Machine Learning (ICML)*, 2019.

[65] C. Merkwirth and T. Lengauer, Automatic generation of complementary descriptors with molecular graph networks. *J. Chem. Inf. Model.* **45** (2005), no. 5, 1159–1168.

[66] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proc. AAAI Conference on Artificial Intelligence*, 2019.

[67] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *Int. Conf. on Learning Representations (ICLR)*, 2019.

[68] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, Relational pooling for graph representations. In *Int. Conference on Machine Learning (ICML)*, 2019.

[69] M. Naor and L. J. Stockmeyer, What can be computed locally? In *Symposium on Theory of Computing (STOC)*, 1993.

[70] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[71] O. Puny, H. Ben-Hamu, and Y. Lipman, From graph low-rank global attention to 2-FWL approximation. In *Int. Conference on Machine Learning (ICML)*, 2020.

[72] C. R. Qi, H. Su, K. Mo, and L. G. PointNet, Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[73] S. Ravanbakhsh, J. Schneider, and B. Poczos, Deep Learning with Sets and Point Clouds. 2016, arXiv:1611.04500v3.

[74] S. Ravanbakhsh, J. Schneider, and B. Póczos, Equivariance through parameter-sharing. In *Int. Conference on Machine Learning (ICML)*, 2017.

[75] R. C. Read and D. G. Corneil, The graph isomorphism disease. *J. Graph Theory* **1** (1977), 339–363.

[76] L. Ruiz, L. F. O. Chamon, and A. Ribeiro, Graphon neural networks and the transferability of graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[77] L. Ruiz, F. Gama, and A. Ribeiro, Graph neural networks: architectures, stability and transferability. *Proc. IEEE* **109** (2021), 660–682.

[78] A. Santoro, F. Hill, D. Barrett, A. Morcos, and T. Lillicrap, Measuring abstract reasoning in neural networks. In *Int. Conference on Machine Learning (ICML)*, pp. 4477–4486, 2018.

[79] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Oascanu, P. Battaglia, and T. Lillicrap, A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[80] R. Sato, M. Yamada, and H. Kashima, Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[81] R. Sato, M. Yamada, and H. Kashima, Random features strengthen graph neural networks. In *SIAM International Conference on Data Mining (SDM)*, 2021.

[82] F. Scarselli, M. Gori, A. Chung Tsoi, M. Hagenbuchner, and G. Monfardini, Computational capabilities of graph neural networks. *IEEE Trans. Neural Netw.* **20** (2009), no. 1, 81–102.

[83] F. Scarselli, M. Gori, A. Chung Tsoi, M. Hagenbuchner, and G. Monfardini, The graph neural network model. *IEEE Trans. Neural Netw.* **20** (2009), no. 1, 61–80.

[84] F. Scarselli, A. C. Tsoi, and M. Hagenbuchner, The Vapnik–Chervonenkis dimension of graph and recursive neural networks. *Neural Netw.* **108** (2018), 248–259.

[85] B. Schölkopf and A. Smola, *Learning with kernels*. Adapt. Comput. Mach. Learn., MIT Press, 2001.

[86] B. Tahmasebi, D. Lim, and S. Jegelka, Counting substructures with higher-order graph neural networks: possibility and impossibility results. 2021, arXiv:2012.03174v2.

[87] A. Trask, F. Hill, S. E. Reed, J. Rae, C. Dyer, and P. Blunsom, Neural arithmetic logic units. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[88] S. M. Ulam, *A collection of mathematical problems*. Interscience Publishers, 1960.

[89] V. N. Vapnik and A. Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.* **16** (1971), no. 2, 264–280.

[90] P. Velickovic, R. Ying, M. Padovano, R. Hadsell, and C. Blundell, Neural execution of graph algorithms. In *Int. Conf. on Learning Representations (ICLR)*, 2020.

[91] S. Verma and Z.-L. Zhang, Stability and generalization of graph convolutional neural networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1539–1548, 2019.

[92] E. Wagstaff, F. B. Fuchs, M. Engelcke, I. Posner, and M. Osborne, On the limitations of representing functions on sets. In *Int. Conference on Machine Learning (ICML)*, 2019.

[93] B. Weisfeiler, *On construction and identification of graphs*. Springer, 1976.

[94] B. Weisfeiler and A. A. Leman, A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia* **2** (1968), no. 9, 12–16.

[95] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, How powerful are graph neural networks? In *Int. Conf. on Learning Representations (ICLR)*, 2019.

[96] K. Xu, J. Li, M. Zhang, S. Du, K. Kawarabayashi, and S. Jegelka, What can neural networks reason about? In *Int. Conf. on Learning Representations (ICLR)*, 2020.

[97] K. Xu, M. Zhang, J. Li, S. Du, K. Kawarabayashi, and S. Jegelka, How neural networks extrapolate: From feedforward to graph neural networks. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

[98] D. Yarotsky, Universal approximations of invariant maps by neural networks. *Constr. Approx.* (2021).

[99] G. Yehudai, E. Fetaya, E. Meirom, G. Chechik, and H. Maron, From local structures to size generalization in graph neural networks. In *Int. Conference on Machine Learning (ICML)*, 2021.

[100] K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, and J. Tenenbaum, Neural-symbolic VQA: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[101] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, Deep sets. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[102] Q. Zhao, Z. Ye, C. Chen, and Y. Wang, Persistence enhanced graph neural network. In *Proc. Int. Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

STEFANIE JEGELKA

Department of EECS, MIT, Cambridge, USA, stefje@mit.edu