Foundations of Graph Neural Networks (A Logician's View)

Egor V. Kostylev
University of Oslo, Norway

Abstract

Graph Neural Networks (GNNs) are a family of neural architectures that are naturally suited to learning functions on graphs. They are now used in a wide range of applications. It has been observed that GNNs share many similarities with classical computer science (CS) formalisms, such as the Weisfeiler-Leman graph isomorphism test, bisimulation, and logic. Most notably, both GNNs and these formalisms deal with functions on graphs and graph-like structures. This observation opens up an opportunity to compare GNN architectures with these formalisms in terms of different kinds of expressibility, thus positioning these architectures within the well-established landscape of theoretical CS. This, in turn, helps us better understand the fundamental capabilities and limitations of various GNN architectures, enabling more informed choices about which architecture to use – if any at all. In these lecture notes, I give an introduction to the state-of-the-art foundations of GNNs – specifically, our current understanding of their expressibility in terms of the classical formalisms, considering several notions of expressive power.

2012 ACM Subject Classification Theory of computation → Query learning; Theory of computation → Modal and temporal logics; Information systems → Query languages

Keywords and phrases Graph Neural Networks, Expressivity, Logic

Digital Object Identifier 10.4230/OASIcs.RW.2024/2025.3

Category Invited Paper

Funding The Research Council of Norway through its Centres of Excellence scheme, Integreat – Norwegian Centre for knowledge-driven machine learning, project 332645.

1 Introduction

Logic-minded people love graphs: they are a very nice and neat data model, useful both in theory and practice. In recent years, machine learning (ML) researchers have also grown fond of graphs: they have realised that usual neural networks (NNs) are insufficient for (abundant) problems where data objects are naturally graph-structured and invented *Graph Neural Networks* (*GNNs*) [16, 7]. Nowadays, graph learning with GNNs is a hot topic in the ML community. There are hundreds, if not thousands, of papers published every year on the topic, where GNNs are studied as a formalism and applied to many problems (where they should and should not), from biology [5] and chemistry [14] to recommender systems [21] and data management [17].

These lecture notes have two main goals. The first is to understand what GNNs are, with a focus on their formalisation and foundations. We will see that GNNs are actually a rich family of formalisms rather than a single architecture, with new GNN-based architectures regularly emerging in the literature. The second goal is to understand what these architectures can and cannot do – that is, to understand the fundamental capabilities and limitations of various GNN architectures, which, in turn, helps us position these architectures within the well-established landscape of theoretical CS, allowing practitioners to make more informed choices about which architecture to use – if any at all. To this end, we will compare these GNN architectures with classical CS formalisms, such as graph isomorphism tests, bisimulations,

and various logics, in terms of their expressive power. This comparison is justified by the fact that, despite the significant differences in their first appearance, GNNs share significant similarities with the classical formalisms, far beyond the basic idea that they deal with graphs. For this comparison, however, it is important to keep in mind that this remains a highly active research area, with several deep and interesting papers emerging each year. Unfortunately, the area is currently subject to considerable hype: many lower-quality papers also appear, making it difficult to identify the truly valuable contributions. Moreover, the terminology has also not yet stabilised, and understanding the relationships between different results often requires delving into technical details. One such example, which is central for our study, is the notion of *expressive power* for GNNs. In particular, we will see that this term is used for several related, but essentially different concepts, which, to avoid ambiguity, we refer to as *distinguishability* (or *separating*), *uniform expressive*, *non-uniform expressive*, and *approximation power*. We will discuss each of these for GNNs, covering the definitions, interrelations, and key results that compare various GNNs to the classic formalisms in terms of each notion of expressivity. Towards the end, we will discuss the limitations of current results and the open problems that can help us complete our understanding of the foundations of GNNs. However, these notes should not be seen as a comprehensive survey of the state of the art, but rather a gentle introduction to the topic.

In these notes, I assume that the reader is familiar with *First Order Logic* (*FOL*). Knowledge of modal or description logics, as well as database query formalisms, such as (unions of) conjunctive queries, is also desirable, but I will introduce them in terms of FOL. Familiarity with GNNs themselves is also beneficial, but if the reader knows too much, then these notes may be less interesting for them.

2 Motivation and Formalisation of Graph Neural Networks

2.1 Graphs and Graph Learning

We begin the formal part with the basic definitions relevant to the notes.

▶ **Definition 1.** *A* graph $G = (V, E, \lambda)$ *with dimension* $d \in \mathbb{N}$ *consists of*

- *a set* V *of nodes,*
- *a set* E *of undirected edges – that is, unordered pairs of distinct elements in* V*, and*
- *a node labelling* $\lambda : V \to \mathbb{R}^d$*.*

Observe that, according to this definition, the graphs are simple and undirected, and the edges are not labelled with numbers or colours (i.e., types). These assumptions are in most cases just for simplicity (and are customary in ML), and many of the results we will discuss generalise to other graphs.

Overall, the objective of graph learning is to learn graph or node embeddings (which are partially or fully unknown). Next, we define such embeddings.

▶ **Definition 2.** *Let* $\mathcal{G}_d$ *be the class of all graphs of some dimension* $d \in \mathbb{N}$ *and* $\mathbb{Y}$ *be an arbitrary set, called an* output space*. Then, a* graph embedding *is a function* $\xi : \mathcal{G}_d \to \mathbb{Y}$*.*

Typical examples of the output space are $\{yes, no\}$ (in which case we can call the embedding *binary classification*), a finite set of classes (*multi-class classification*), and real numbers (*regression*). A common example of a practical graph embedding learning task with binary classification is predicting whether a molecule, represented as a graph, possesses a certain property, such as toxicity.

▶ **Definition 3.** *Let $\mathcal{G}_d$ be the class of all graphs of some dimension d, $\mathcal{GV}_d$ be the class of all pairs (G, v), where $G = (V, E, \lambda) \in \mathcal{G}_d$ and $v \in V$, and $\mathbb{Y}$ be an output space. Then, a* node embedding *is a function $\xi : \mathcal{GV}_d \to \mathbb{Y}$.*

An example of a node embedding learning task is predicting the relevance of a paper in a citation network (for a particular user) – that is, to assign a number in $[0, 1]$ for every paper. In these notes, we mostly focus on node embeddings, but our results can often be adapted to whole graphs.

In passing, it is worth noting that, in graph learning literature, graphs are often represented as two matrices: the adjacency matrix of the graph structure and the labelling matrix that stacks the labels of the nodes. However, such a representation becomes dependent on the order of the nodes, which is not specified in the graphs and should therefore be fixed in some way. In this case, it is usually required for graph embeddings and node embeddings (which now take matrix representations as input instead of graphs themselves) to be *invariant* and *equivariant*, respectively – that is, be independent of the order of nodes in the representation.

2.2 What Is Special About Graph Learning?

An immediate question after these definitions is why standard deep neural networks and other traditional ML formalisms cannot be used for graph learning. To answer this, let us first recall that a (fully-connected feed-forward) *neural network* (*NN*) consists of multiple layers of so-called neurons, where each neuron in each layer is connected to each neuron in the next layer with an assigned (learnable) numeric weight. The number of neurons in each layer is called its *dimension*. Such an NN realises a function mapping vectors of real numbers of the first layer dimension to such vectors of last layer dimension by feeding the input vector to the input layer neurons and computing the value $\lambda(n)$ of each non-input neuron n as $f(\sum_{n'} w_{n' \to n} \cdot \lambda(n'))$, where n' ranges over the neurons of the previous layer, $w_{n' \to n}$ is the weight assigned to the connection from n' to n, and f is a non-linear function, such as sigmoid or ReLU. This architecture can be easily adapted for non-numeric outputs, such as classification, by applying an additional fixed function with appropriate range, for example, softmax, to the last-layer vector. Usually, the connection weights are obtained through (supervised) *training*, which optimises them via backpropagation to minimise the loss function – that is, the difference between the NN's predictions and the ground-truth answers for known examples.

However, applying such (fully connected) NNs for graph learning faces several problems. The first one is scalability: when a layer has significantly many neurons, there can be prohibitively many weights. For example, for a rather reasonable $250 \cdot 250$ grid graph (i.e., the graph with nodes (i, j), for $i, j \in \{1, \ldots, 250\}$, and an edge between (i, j) and $(i + 1, j)$, and between (i, j) and $(i, j + 1)$ for all admissible i, j) with dimension 1 as input and just 500 neurons per intermediate layer we already require at $250 \cdot 250 \cdot 500 = 31,250,000$ weights between the first two layers (assuming that the graph is represented as a vector by concatenating its rows). The second problem is that the inputs to an NN are always totally ordered, and so the same graph may be represented as an NN's input vector in many ways. As a result, NNs do not respect graph symmetries – that is, may give different answers for the same node, depending on the graph's representation; teaching an NN these symmetries often requires additional, sometimes prohibitive, training effort. The third, and more fundamental, problem is that each NN requires fixed-size input, whereas graph learning does not assume any restriction on the size of the input graph; in particular, a model trained on small graphs should be applicable to graphs of arbitrarily large size. While one could impose a hard bound on graph sizes, this is, of course, not a principled solution.

For grid graphs, a possible solution is to use *Convolutional Neural Networks* (CNNs) – a well-established neural approach widely used in, for example, image processing. The key idea behind CNNs is that all layers maintain the same number of neurons, and each neuron has connections only from a small *window* (i.e., neighbourhood) of neurons in the previous layer, centred on the corresponding neuron in that layer. Moreover, the weights are now shared across the windows – that is, are determined by the relative positions of the connected neurons in the grid (and the layer number) rather than by absolute positions. This drastically reduces the number of weights and allows CNNs to handle grids of arbitrary size. Thus, in essence, CNNs are designed to recognise local patterns.

As we will see next, GNNs generalise the idea of CNNs to arbitrary graphs.

2.3 Graph Neural Networks

Overall, GNNs are a family of ML architectures that leverage the structure of graphs to effectively learn graph and node embeddings. Following Gilmer et al. [7], who first noticed that many architectures for graph learning have a common pattern, and Barcelo et al. [3], we next give a formalisation of GNNs in a general abstract form, which serves as an umbrella for many GNN architectures. We begin with the syntax of GNNs for node embeddings.

▶ **Definition 4.** *A* node-level aggregate-combine GNN *(*AC-GNN*) with L layers, for $L \in \mathbb{N}$, input dimension $d \in \mathbb{N}$, (hidden) dimensions $d_1, \ldots, d_L$ all in $\mathbb{N}$, and output space $\mathbb{Y}$ is the following two families of functions and one single function, where $d_0 = d$ and, as usual, $\mathbb{N}^{\mathbb{R}^{d_{\ell-1}}}$ is the set of all multisets of real vectors of size $d_{\ell-1}$:*

- *aggregation* $\mathrm{AGG}^{(\ell)} : \mathbb{N}^{\mathbb{R}^{d_{\ell-1}}} \to \mathbb{R}^{d_\ell}$*, for every* $\ell \in \{1, \ldots, L\}$*,*
- *combination* $\mathrm{COMB}^{(\ell)} : \mathbb{R}^{d_{\ell-1}+d_\ell} \to \mathbb{R}^{d_\ell}$*, for every* $\ell \in \{1, \ldots, L\}$*, and*
- *output* $\mathrm{OUT} : \mathbb{R}^{d_L} \to \mathbb{Y}$*.*

In what follows, for brevity, in the case of $\mathbb{Y} = \mathbb{R}^{d_L}$ and $d_L = 1$ (e.g., regression) we can take OUT to be identity and omit it. We now proceed with the semantics.

▶ **Definition 5.** *A node-level AC-GNN as in Definition 4 runs on an input graph $G = (V, E, \lambda)$ of dimension d by*

- *initialising* $\boldsymbol{x}_v^{(0)} \coloneqq \lambda(v)$ *for every* $v \in V$*,*
- *iteratively computing* $\boldsymbol{x}_v^{(\ell)} \in \mathbb{R}^{d_\ell}$ *for each* v *and* $\ell \in \{1, \ldots, L\}$ *as follows, where* $\mathcal{N}_G(v)$ *is the set of neighbours of* v *in* G *and* $\{\{\cdot \mid \cdot\}\}$ *is the multiset constructor:*

$$\boldsymbol{x}_v^{(\ell)} \coloneqq \mathrm{COMB}^{(\ell)}\left(\boldsymbol{x}_v^{(\ell-1)}, \mathrm{AGG}^{(\ell)}\left(\{\{\boldsymbol{x}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\}\right)\right). \tag{1}$$

Then, the result of the AC-GNN on G is $\mathrm{OUT}(\boldsymbol{x}_v^{(L)})$ *for each* $v \in V$*.*

Thus, node-level AC-GNNs realise node embeddings. Since we focus on node embeddings in these notes, we will usually omit "node-level" when referring to AC-GNNs. For completeness, however, we also briefly describe how to adapt AC-GNNs to the graph level. In particular, the syntax of *graph-level AC-GNNs* is the same as in Definition 4 except it includes the output function $\mathrm{READOUT} : \mathbb{N}^{\mathbb{R}^{d_L}} \to \mathbb{Y}$ instead of OUT; moreover, the semantics is the same as in Definition 5 except that there is one result for the entire graph, computed as $\mathrm{READOUT}(\{\{\boldsymbol{x}_v^{(L)} \mid v \in V\}\})$. Thus, graph-level AC-GNNs realise graph embeddings.

Our definition of the AC-GNN family of architectures is abstract; in fact, it does not have anything "neural" in it by itself. However, it covers most of concrete GNN architectures used in practice in graph learning, and which therefore have trainable weights. In particular, in

these GNNs, the functions $\mathrm{AGG}^{(\ell)}$ and $\mathrm{COMB}^{(\ell)}$ rely on matrices and vectors of numbers, as well as on some non-linearity. There are many such architectures, and even an overview of them is beyond the scope of these notes. Instead, we limit ourselves to basic examples.

▶ **Example 6.** Basic *Sum-Plus GNNs with sigmoid* instantiate equation (1) as

$$\boldsymbol{x}_v^{(\ell)} \coloneqq \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right), \tag{2}$$

where

- the $\boldsymbol{A}^{(\ell)}$ and $\boldsymbol{C}^{(\ell)}$ are numeric matrices of weights of appropriate dimensions,
- the $\boldsymbol{b}^{(\ell)}$ are numeric vectors of weights of appropriate sizes, and
- $\mathit{Sigmoid}$ is the sigmoid function defined as $\mathit{Sigmoid}(x) = 1/(1+e^{-x})$ (and applied element-wise to vectors).

Observe that this is indeed an *AC-GNN*: $\mathrm{AGG}^{(\ell)}$ is the sum and $\mathrm{COMB}^{(\ell)}$ is everything else.

At this point, it is worth noticing a common abuse of terminology: when referring to "a GNN" ("AC-GNN", "Sum-Plus GNN with sigmoid", etc.), we may mean either a concrete instance specified by given aggregation, combination, etc. functions, or a family of such instances (also known as a *GNN architecture*), which is defined by restricting the functions to certain classes without specifying concrete members of these classes. For example, a sum-plus GNN with all matrix and vector weights given (e.g., trained) is an instance, but an untrained sum-plus GNN, where only the number of layers, the dimensions, and the structure of formula (2) are specified, is an architecture. Usually, this abuse does not cause any confusion, but we will specify what we mean if there is any risk of ambiguity.

▶ **Example 7.** In equation (2) of Example 6, we can have

- element-wise average (denoted as avg below), max, etc. instead of the sum as aggregation,
- vector concatenation $||$, etc. instead of $+$ and ReLU, etc. instead of sigmoid in combination,

where, for each $x \in \mathbb{R}$,

$$\mathit{ReLU}(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{otherwise.} \end{cases}$$

Other common examples of GNNs include (a version of) *Graph Convolutional Networks (GCNs)* [17], which instantiate equation (1) as

$$\boldsymbol{x}_v^{(\ell)} \coloneqq \mathit{ReLU}\left(\boldsymbol{A}^{(\ell)}\left(\mathrm{avg}_{u\in\mathcal{N}_G(v)\cup\{v\}}\boldsymbol{x}_u^{(\ell-1)}\right)\right),$$

and *GraphSAGE* [9], instantiating it as

$$\boldsymbol{x}_v^{(\ell)} \coloneqq \boldsymbol{A}^{(\ell)}\left(\boldsymbol{x}_v^{(\ell-1)} \,\middle|\middle|\, \left(\max\nolimits_{u\in\mathcal{N}_G(v)} \mathit{ReLU}(\boldsymbol{C}^{(\ell)}\boldsymbol{x}_u^{(\ell-1)})\right)\right),$$

where the $\boldsymbol{A}^{(\ell)}$ and $\boldsymbol{C}^{(\ell)}$ are again (trainable) matrices of appropriate dimensions.

Note, however, that many recent advanced GNN architectures do not fall into the AC-GNN framework. In fact, most such architectures have been developed as attempts to address the expressivity limitations of AC-GNNs, many of which were identified in the papers I will refer to below. Among these limitations is, for example, the fact that the node embedding "the node is reachable from a node with label 1" is not expressible by any AC-GNN, as each AC-GNN has a fixed number of layers. *Recurrent GNNs* [13] are, in principle, capable of learning (i.e., expressing) such reachability. Another example of a limitation is that the

node embedding "the node is on a cycle of length 5" is also not expressible by any AC-GNN. However, *5-WL GNNs* [11] can, in principle, detect a 5-cycle. Finally, the node embedding "the node is such that there is another node with label 1 somewhere in the graph" is also not expressible by any AC-GNN, but *GNNs with global readouts* [3] can look globally, even in other connected components, in the input graph. In these notes, we primarily focus on *AC-GNNs* and briefly discuss more powerful architectures at the end.

So, we intuitively see that different GNN architectures (within or outside the AC-GNN framework) can do some things and cannot do others. However, before studying this issue in a principled way, we need to understand what this formally means. Based on the state of the art, we next consider several related notions for GNNs, each sometimes called *expressive power*: distinguishing (or separating) power (Section 3), (uniform) expressive power (Section 4), non-uniform expressive power (Section 5), and (non-uniform) approximation power (also Section 5). Understanding expressive power (for different notions of expressibility) may reveal, for example, the following: which information in graphs is used by a given GNN-based embedding method, which embeddings could – in principle – be learned by this method, and which more powerful methods may be needed for the application at hand. However, before proceeding to our first notion, we emphasise that "weaker expressivity" (of any kind) does not mean "worse," since it just presupposes more *a priori* knowledge, which is embedded in the GNN architecture and hence does not need to be re-learned from examples.

3 Distinguishing Power of GNNs

We begin with distinguishing (also referred to as separating) power, which was historically the first notion of expressibility in the context of GNNs; it was introduced independently by Morris et al. [11] and Xu et al. [20] in 2019. Intuitively, a GNN architecture can distinguish at least as much as another formalism of node embeddings if, for every pair of nodes in graphs (same or different) with different answers given by some node embedding in the formalism, there is a GNN within the architecture that also gives different answers for this pair of nodes.

▶ **Definition 8.** *A class $\mathcal{C}_2$ of node embeddings has at least as* strong distinguishing (or separating) power *as a class $\mathcal{C}_1$ when $\xi_1(G, v) \neq \xi_1(G', v')$ for some G, v, G', v', and $\xi_1 \in \mathcal{C}_1$ implies that $\xi_2(G, v) \neq \xi_2(G', v')$ for some $\xi_2 \in \mathcal{C}_2$. Classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the* same distinguishing power *if $\mathcal{C}_1$ is also at least as strong as $\mathcal{C}_2$.*

Note that this definition is relevant even when one of $\mathcal{C}_1, \mathcal{C}_2$ is a singleton. Also, note that even if the property of this definition is formulated for classes of embeddings, it naturally extends to formalisms realising embeddings, such as AC-GNNs. We next state our first (and now well-known) result by by Morris et al. [11] and Xu et al. [20].

▶ **Theorem 9.** *AC-GNNs have the same distinguishing power as the stable colouring of the Weisfeiler-Leman graph isomorphism (WL) test.*

To make sense of this result, we next introduce the WL test [19] and then give an idea of the proof of this theorem.

3.1 Weisfeiler-Leman Graph Isomorphism Test

Before we move to the definition, we briefly discuss the role of the test in CS. First, the WL test [19] is a classical formalism, so it is a useful reference point for distinguishability. For example, it is well-known that, on the graph level, it has the same distinguishing power as *FOL with two variables and counting* (C^2), and its node level version – that is, the stable

colouring, which is relevant to Theorem 9, – as *graded modal logic* (we will come back to this logic in Section 4.2). However, the main property of this test is that it is a complete yet not sound test for isomorphism between two graphs:

- if it rejects, then the graphs are not isomorphic, but
- if it accepts, then they may be isomorphic or not.

This test is useful, because, on the one hand, the answer "accept" for non-isomorphic graphs is possible for only special, practically rare cases, and, on the other hand, the complexity of checking isomorphism of two graphs is likely intractable (to be more precise, this is a big open problem in CS: neither polynomial algorithm nor NP-hardness proof is known).

As we will see, the WL test works with graphs where nodes are assigned with colours, which is, in principle, different from the graphs with numeric labellings as in Definition 1. However, this mismatch is not essential for us: in this section, we can assume that each numeric vector has a unique corresponding colour.

▶ **Definition 10.** *The* Weisfeiler-Leman (WL) graph isomorphism test *(also called* colour refinement*) takes two graphs with coloured nodes as input and iterates the following until the colouring is* stable *– that is, the partition of the nodes into colours does not change:*

- *two nodes v, v' are assigned the same colour at an iteration if and only if they have the same colour and the same multisets of colours of neighbours before the iteration.*

Then, the answer is "accept" if the two graphs have the same multiset of colours of all their nodes and "reject" otherwise.

Note that the colouring always stabilises, as each iteration refines the colouring of the previous one. Figure 1 depicts two examples of the WL test applied to pairs of graphs (where the nodes in the input graphs are all of the same colour, which is just the property of these examples, but not required in the definition). In Figure 1a, the stable colourings for the two graphs are different and so the test rejects, while in Figure 1b they are the same and so it accepts; however, in both cases the graphs are not isomorphic.

3.2 AC-GNNs and Weisfeiler-Leman Test

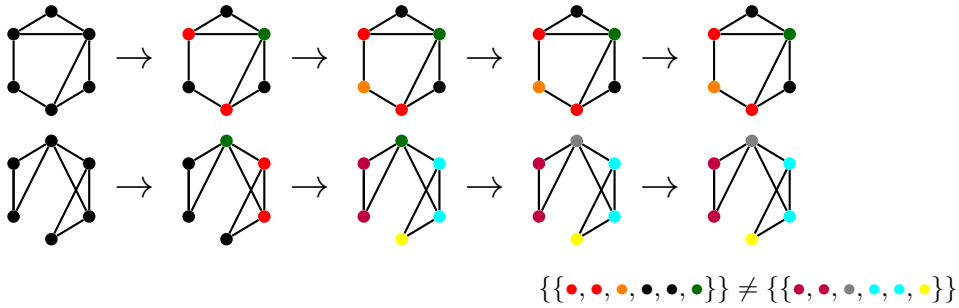
Thus, for every node v in every graph G, the colour $\mathrm{WL}_v^{(\ell)}$ of v after ℓ iterations of the WL test is given by the following, where each $\mathrm{INJ}^{(\ell)}$ is an injective function mapping a colour and a multiset of colours to another colour:

- $\mathrm{WL}_v^{(0)} \coloneqq \lambda(v)$,
- $\mathrm{WL}_v^{(\ell)} \coloneqq \mathrm{INJ}^{(\ell)}(\mathrm{WL}_v^{(\ell-1)}, \{\{\mathrm{WL}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\})$.

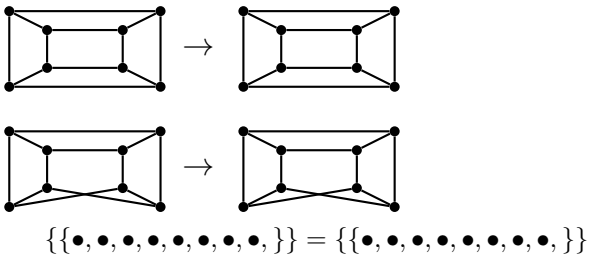
As we know, an AC-GNN for each such node v works as follows:

- $\boldsymbol{x}_v^{(0)} \coloneqq \lambda(v)$,
- $\boldsymbol{x}_v^{(\ell)} \coloneqq \mathrm{COMB}^{(\ell)}(\boldsymbol{x}_v^{(\ell-1)}, \mathrm{AGG}^{(\ell)}(\{\{\boldsymbol{x}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\}))$.

In other words, the WL test works exactly as an AC-GNN with injective aggregation and combination functions – that is, if WL assigns the same value to two nodes at iteration ℓ, then every AC-GNN also assigns the same value to these two nodes at layer ℓ. Moreover, we can say that the WL test realises a node embedding (or a class of embeddings, but all with the same distinguishing power) that assigns to each node the colour of the stable colouring. This correspondence is the essence of the proof of Theorem 9: the "WL simulates AC-GNNs" direction follows from the injectiveness, while for the "GNNs simulate WL" direction, we just need to choose the number of layers sufficiently large so that the WL stabilises on the two given graphs (recall that the definition of distinguishability allows for a separate GNN for every two graph-node pairs).



(a) Colour partitioning stabilises after four iterations; initially and after the first iteration the corresponding two multisets of colours are the same, but after the second iteration and till the stabilisation the multisets are different; hence the WL test outputs "reject" (which is correct: the graphs are not isomorphic).



(b) The initial colour partitioning is already stable; final colourings are the same, and hence the WL test outputs "accept" (which is wrong: the graphs are not isomorphic).

Figure 1 WL colouring refinement for two pairs of graphs, where the pairs graphs on the left are initial graphs with coloured nodes (all black in these cases) and parallel arrows represent iterations of the colour refinement; note that the same colour in different iterations is not meaningful: WL concerns itself only with the colours in the same iteration.

Let us now return to the fact that WL works as an AC-GNN with injective $\mathrm{AGG}^{(\ell)}$ and $\mathrm{COMB}^{(\ell)}$ functions. Such functions obviously exist. However, the immediate question is whether they can be truly "neural," in a manner similar to our Sum-Plus GNNs. We can try

$$\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} \;\Big\|\; \boldsymbol{C}^{(\ell)}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} f(\boldsymbol{x}_u^{(\ell-1)})\right)\right) \tag{3}$$

for some smart f such that $\sum_{x\in X} f(x)$ is always injective. Unfortunately, it can be proven that such an f that remains injective for arbitrarily large X does not exist. Luckily, we can use the idea that enabled us to capture, in terms of distinguishability, the WL test with non-fixed number of iterations until stabilisation with AC-GNNs with fixed number of layers: we do not need X to be arbitrarily large, but only as large as the number of nodes in the input graphs, thus ensuring that $\mathrm{AGG}^{(\ell)}$ is injective for these graphs (though not for all graphs). Such a function f exists, and can be constructed using a straightforward generalisation of so-called *deep sets* [22]. The resulting GNN is essentially the so-called *GIN*[1] [20].

We have started this section by stating that distinguishability is historically the first and most popular notion of expressibility for GNNs. However, is it really adequate for our understanding of what GNNs can and cannot do? We contend that this may not be the case. Indeed, we observe that

[1] This stands for "Graph Isomorphism Network," though this name is somewhat misleading, as their relevance to graph isomorphisms is debatable.

- the WL test realises a node embedding,
- each AC-GNN realises a node embedding, and
- AC-GNNs have the same distinguishing power as WL.

However, this does not guarantee that there is an AC-GNN that gives the same answer as the WL test for every node in every graph; in fact, this claim is obviously incorrect, since each AC-GNN has a fixed number of layers, whereas the WL test may require an arbitrarily large number of iterations, depending on the graph. To compare GNNs with other formalisms from this perspective we need a stronger, uniform notion of expressivity.

4 Uniform Expressive Power of GNNs

The following definition of "true," uniform expressive power is stronger than the one based on distinguishability. It is also common in theoretical computer science (in particular, in logic), and was first introduced to the context of GNNs by Barcelo et al. [3] in 2019.

▶ **Definition 11.** *A class $\mathcal{C}_2$ of node embeddings has at least as* strong (uniform) expressive power *as class $\mathcal{C}_1$ if $\mathcal{C}_1 \subseteq \mathcal{C}_2$. Classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the* same expressive power *if $\mathcal{C}_1 = \mathcal{C}_2$.*

This notion, rather straightforward by itself, again naturally extends to formalisms realising node embeddings, when it becomes more interesting. We start with expressivity of AC-GNNs in terms of a special kind of bisimulation (which is in essence a modification of Theorem 9 about distinguishability). Then, at last, we look at connections to logic.

4.1 Expressivity via Bisimulation

Bisumulation is a classical notion that appears in many areas of CS: verification, databases, modal logics, etc. Intuitively, a bisimulation is a relation between nodes of two graphs showing that they can match each other's moves along edges step by step, so that the nodes are behaviourally indistinguishable. We next give a standard definition followed by a stronger version that takes the number of successors into account, which is more relevant for GNN expressibility.

▶ **Definition 12.** *Let $G_1 = (V_1, E_1, \lambda_1)$ and $G_2 = (V_2, E_2, \lambda_2)$ be graphs. Relation $\rho \subseteq V_1 \times V_2$ is a* bisimulation *if, for each pair $(v_1, v_2) \in \rho$,*

- *$\lambda_1(v_1) = \lambda_2(v_2)$,*
- *for every $v_1' \in \mathcal{N}_{G_1}(v_1)$ there is $v_2' \in \mathcal{N}_{G_2}(v_2)$ with $(v_1', v_2') \in \rho$, and*
- *for every $v_2' \in \mathcal{N}_{G_2}(v_2)$ there is $v_1' \in \mathcal{N}_{G_1}(v_1)$ with $(v_1', v_2') \in \rho$.*

Bisimulation ρ is a counting bisimulation *when v_2' and v_1' in the latter two conditions is distinct for each neighbour v_1' and v_2', respectively, of v_1 and v_2 – that is, when the number of bisimilar neighbours of v_1 and v_2 in each counting bisimulation class is the same.*

Note that, in this definition, graphs G_1 and G_2 may be the same. Another observation is that a union of two usual or counting bisimulations is also such a bisimulation, so we can refer to the *maximal bisimulation* and the *maximal counting bisimulation*. Notably, the same-colouring relation of the result of the WL test is the maximal counting bisimulation.

As we may see, counting bisimulation (same as the usual one) may depend on nodes arbitrary far from each other in the graph. The following partial version (in fact, a family of versions) looks only up to a fixed depth, making it an approximation of the general notion.

▶ **Definition 13.** *Let $G_1 = (V_1, E_1, \lambda_1)$ and $G_2 = (V_2, E_2, \lambda_2)$ be graphs. Relation $\rho_\ell \subseteq V_1 \times V_2$ is a* ℓ-partial counting bisimulation*, for $\ell \in \mathbb{N}$, if, for each pair $(v_1, v_2) \in \rho_\ell$,*

- $\lambda_1(v_1) = \lambda_2(v_2)$,
- *for every $v_1' \in \mathcal{N}_{G_1}(v_1)$ there is a distinct $v_2' \in \mathcal{N}_{G_2}(v_2)$ with $(v_1', v_2') \in \rho_{\ell-1}$, and*
- *for every $v_2' \in \mathcal{N}_{G_2}(v_2)$ there is a distinct $v_1' \in \mathcal{N}_{G_1}(v_1)$ with $(v_1', v_2') \in \rho_{\ell-1}$;*

here, for uniformity, we assume that ρ_0 is defined in the same way as the other ρ_ℓ except that only the first condition should hold.

For example, the same-colouring relation of the intermediate result of WL after ℓ iterations is the maximal ℓ-partial counting bisimulation (which exists and is unique for the same reasons as in the general case).

Every type of bisimulation corresponds to a specific class of node embeddings.

▶ **Definition 14.** *A node embedding ξ is* (ℓ-partial, counting) bisimulation invariant *if $\xi(G, v) = \xi(G', v')$ for every pair (v, v') of nodes in graphs G_1 and G_2 in the maximal (ℓ-partial, counting, respectively) bisimulation.*

In other words, no invariant embedding can separate bisimilar nodes. It is immediate that every ℓ-partial counting bisimulation invariant embedding is also $(\ell+1)$-partial counting bisimulation invariant. Thus, we have the following:

- every AC-GNN with L layers realises an L-partial counting bisimulation invariant embedding, since both care about only tree-unravellings (i.e., infinite trees rooted at the nodes whose branches correspond to all starting from the nodes);
- every L-partial counting bisimulation invariant embedding is realised by an AC-GNN with L layers, since we can take injective aggregation and combination and make the final output function do all the job.

These observations give rise to the following semantic characterisation of uniform expressivity of AC-GNNs by Pflueger et al. [13].

▶ **Theorem 15.** *AC-GNNs is equivalent in expressive power to the class of all embeddings each of which is ℓ-partial counting bisimulation invariant for some ℓ.*

Observe that even though ℓ is not bounded in this theorem, "partial" is essential: there exist counting bisimulation invariant embeddings that are not realised by any AC-GNN. It is also not surprising that we may need an AC-GNN with functions of arbitrary complexity (e.g., not even computable) to express some (arbitrarily complex) invariant functions. Thus, there may not be any "real" GNN – that is, of form (2) or similar – that expresses many such functions. However, it is a good upper bound for all subclasses of AC-GNNs.

4.2 Expressivity via Logic: Graded Modal Logic Case

The semantic characterisation of Theorem 15 opens up a possibility to compare (uniform) expressibility of GNNs with those of logical formalisms. Logics are fundamental in many areas of computer science, and we have a very fine-grained landscape of logics in terms of expressive power. Therefore, it could be beneficial to leverage this knowledge for GNNs. Comparing GNNs and logics is very natural: they have a lot in common, since both realise functions with graph-like domains. For example,

- node-level GNNs realise node embeddings – that is, functions from graph-node pairs to output sets, and
- unary logic (e.g., FOL) formulas realise functions from structure-element pairs to $\{yes, no\}$.

A similar correspondence exists for graph-level GNNs and logical sentences.

However, even if both logic and GNNs work with similar objects, they are not exactly the same in their full generality. Therefore, we first need to find common ground – that is, unify the input and output spaces. For the input, we observe that every finite logical structure over the signature with one symmetric binary relation E and several unary relations can be seen as a graph in the GNN sense: we just need to assign a dedicated position in the embedding vector to each unary predicate and use the 1-0 encoding of what holds in the structure; for example, if $A(a)$ and $C(a)$ hold in a structure for the signature with unary predicates A, B, C, then this can be seen as the label $(1, 0, 1)$ of node a in the graph. For the output, we can restrict our attention to AC-GNNs with output space $\{yes, no\}$ – that is, to binary node classification. Note that real-life instances of AC-GNNs (Sum-Plus GNNs, GraphSAGE, GCNs, etc.) can be converted to such by, for example, a threshold as the output function. For the rest of the paper, we usually silently assume only such inputs (i.e., logical structures and, equivalently, GNN graphs with 1-0 labellings) and outputs (i.e., $\{yes, no\}$ as in binary classification), and we will say explicitly when it is not the case.

As we know from Theorem 15, AC-GNNs can express functions far beyond any usual logic. In fact, the families of node classifiers realised by common practical GNN architectures (GCNs, GraphSAGE, etc.) can also express a lot outside usual logics due to sophisticated aggregation over neighbours, and, especially, non-linearity (sigmoid, ReLU, etc.). There are results that identify (very expressive) logics capturing, in terms of expressive powers, some of such architectures; we will come back to them later, in Section 4.4. However, our first goal is to focus on all AC-GNN-based classifiers that are also expressible in FOL (with equality). In particular, our next result, which is due to Barcelo et al. [3], is the following.

▶ **Theorem 16.** *A node classifier is realisable by both an AC-GNN and an FOL formula if and only if it is realisable by a graded modal logic formula.*

We will sketch the proof of this result, where we will, for the backward direction, reduce to a neural subclass of AC-GNNs. Before this, however, we need to briefly introduce the *graded modal logic* (*GML*). In fact, it is a fragment of FOL with its own syntax (and terminology):

$$\phi ::= p \mid \phi \wedge \phi \mid \neg\phi \mid \Diamond_n \phi.$$

If the reader is more familiar with description logics, they may know this fragment as $\mathcal{ALCQ}$ with one role name E, also with its own syntax (and terminology):

$$C ::= A \mid C \sqcap C \mid \neg C \mid \exists^n E.C.$$

However, as mentioned above, it is well known that we can see GML as the syntactic fragment of unary FOL specified by the following grammar, with the semantics inherited from FOL, and it will be convenient for us to assume this fragment when referring to GML:

$$\varphi(x) ::= A(x) \mid \varphi(x) \wedge \varphi(x) \mid \neg\varphi(x) \mid \exists^{\geq n} y.\, (E(x, y) \wedge \varphi(y)).$$

In this syntax, we used counting quantifiers $\exists^{\geq n}$, which are syntactic sugar in FOL, since they are expressible via inequalities. This, however, comes with a cost of extra variables, and so FO^2, FOL with two variables, is strictly less expressive than C^2, FOL with two variables and counting quantifiers. As we can immediately see from the definition, GML is inside C^2 but not in FO^2.

We are ready to give an idea of the proof of Theorem 16. We start with the backward direction.

▶ **Lemma 17.** *For every GML formula there is a binary AC-GNN node classifier realising the same embedding.*

To prove this lemma, we construct an AC-GNN classifier instantiating equation (1) as

$$\boldsymbol{x}_v^{(\ell)} := trReLU\left(\boldsymbol{A}\,\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}\right),$$

where $trReLU$ is the *truncated ReLU* defined as

$$trReLU(x) = \begin{cases} 0 \text{ if } x < 0, \\ x \text{ if } 0 \leq x < 1, \\ 1, \text{ otherwise,} \end{cases}$$

while $\boldsymbol{A}, \boldsymbol{C}$ are matrices and $\boldsymbol{b}$ is a vector of weights such that each feature vector $\boldsymbol{x}_v^{(\ell)}$ of each node v has one component $\boldsymbol{x}_v^{(\ell)}[\varphi'] \in \{0,1\}$ for each subformula φ' of φ satisfying

- $\boldsymbol{x}_v^{(\ell)}[\varphi_1 \wedge \varphi_2] = trReLU(\boldsymbol{x}_v^{(\ell-1)}[\varphi_1] + \boldsymbol{x}_v^{(\ell-1)}[\varphi_2] - 1)$,
- $\boldsymbol{x}_v^{(\ell)}[\neg\varphi'] = trReLU(-\boldsymbol{x}_v^{(\ell-1)}[\varphi'] + 1)$, and
- $\boldsymbol{x}_v^{(\ell)}[\exists^{\geq n} y.\, E(x,y) \wedge \varphi'(y)] = trReLU(\sum_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}[\varphi'] - (n-1))$.

As a result, for L the structure depth of φ, we have $\boldsymbol{x}_v^{(L)}[\varphi] = 1$ if and only if $v \models \varphi(x)$. Note that the matrices and the vector of the weights – that is, the aggregation and combination functions – are the same for all layers of the GNN. Such GNNs are sometimes called *homogeneous*.

The forward direction of Theorem 16 is claimed by the following lemma.

▶ **Lemma 18.** *For every AC-GNN node classifier that is also in FOL, there is a GML formula realising the same embedding.*

The proof relies on the following facts. First, we know from Theorem 15 that AC-GNN node classifiers are all partial counting bisimulation invariant. Then, as follows immediately from the definitions, every partial counting bisimulation invariant embedding is counting bisimulation invariant. Finally, Van Benthem-style (rather deep) theorem for GML by Otto [12] says that the counting bisimulation-invariant fragment of FOL is precisely GML (for both all and only finite structures, with only the latter relevant for us).

4.3 Expressivity via Logic: Unions of Tree-Shaped Conjunctive Queries

Theorem 16 does characterise a class of GNNs, but this class is not syntactically defined. Thus, an immediate question is whether we can find a (preferably nice-looking) syntactic subclass of AC-GNN classifiers that is equivalent in expressive power to a known logic. For GML (and plain modal logic) it may be hard, or even probably impossible for a reasonable notion of "nice-looking." Our next result, which is due to Tena Cucala et al. [18], achieves this for a simple positive fragment of GML: unions of tree-shaped conjunctive queries (tree-UCQs).

▶ **Theorem 19.** *Monotonic max GNN node classifiers have the same expressive power as unary tree-UCQs.*

As in the previous cases, we begin by defining the logic and the class of GNNs. A *unary tree-shaped conjunctive query* (*tree-CQ*) is an existentially quantified conjunction of unary and binary atoms with one free variable and with the shape of the directed tree with the variable as the root. More precisely, *unary tree-CQs* are a restriction of our FO fragment for GML that adheres to the following grammar:

$$\varphi(x) ::= A(x) \mid \varphi(x) \wedge \varphi(x) \mid \exists y.\, (E(x,y) \wedge \varphi(y));$$

then, a *unary tree-UCQ* is a *disjunction (or union)* of unary tree-CQs with the same free variable. In other words, unary tree-UCQs are the positive, counting-free fragment of GML.

To find a fragment of AC-GNNs that corresponds to unary tree-UCQs, we first observe that each unary tree-UCQ Q is *monotonic under homomorphisms* – that is, is such that for every two graphs $G_1 = (V_1, E_1, \lambda_1)$ and $G_2 = (V_2, E_2, \lambda_2)$ with a function $h : V_1 \to V_2$ for which

- $h(E_1) \subseteq E_2$ (i.e., the edges are preserved), and
- $\lambda_1(v) \leq \lambda_2(h(v))$ for every $v \in V_1$ (i.e., the labellings of the nodes are also preserved with respect to the order)

we have that if $Q(v)$ is true, then $Q(h(v))$ is also true for every $v \in V_1$. However, as the following example shows, not all AC-GNNs are monotonic under homomorphisms.

▶ **Example 20.** Consider a GNN with dimension 3 and one layer that is defined as

$$\boldsymbol{x}_v^{(1)} := ReLU\left(\boldsymbol{A}\boldsymbol{x}_v^{(0)} + \boldsymbol{C}\left(\sum\nolimits_{u \in \mathcal{N}_G(v)} \boldsymbol{x}_u^{(0)}\right) + \boldsymbol{b}\right),$$

where

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & -2 & 0 \end{pmatrix},$$

while $\boldsymbol{C}$ and $\boldsymbol{b}$ consist only of zeroes, and the output classification function is threshold-1 of the third element. Then, for G with a single node v such that $\lambda(v) = (1,0,0)$, the result for v is 1. However, for G' with a single v' such that $\lambda(v') = (1,1,0)$, the result is 0.

Note that negative weights in matrices is not the only problem: sum can "count," but UCQs cannot. Thus, our target fragment of AC-GNNs is also such.

▶ **Definition 21.** A monotonic max GNN *node classifier instantiates equation* (1) *as*

$$\boldsymbol{x}_v^{(\ell)} := \sigma\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\max\nolimits_{u \in \mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right),$$

where

- *all weights in matrices* $\boldsymbol{A}^{(\ell)}$ *and* $\boldsymbol{C}^{(\ell)}$ *are non-negative, and*
- *the activation function* σ *is monotonically increasing, unbounded, and has non-negative range;*

moreover, the output classification function OUT *is a threshold function for some element of the last-layer node labelling.*

It is not difficult to see that the node embedding of a monotonic max GNN is monotonic under homomorphisms. So, we have a hope that Theorem 19 indeed holds. In fact, we can prove the theorem by means of the following two lemmas.

▶ **Lemma 22.** *For every unary tree-UCQ there is a monotonic max GNN node classifier realising the same embedding.*

This lemma can be proven using similar ideas as Lemma 17 for GML. The main differences are that we have to use max instead of sum, and we cannot use the truncated ReLU, since it is bounded, so we have to use the standard ReLU; however, this suffices, since we do not need to simulate negation and counting.

▶ **Lemma 23.** *For every monotonic max GNN node classifier there is a unary tree-UCQ realising the same embedding.*

The proof of this lemma is based on three main ideas.

First, we can check whether a tree-CQ is sound for a monotonic max GNN – that is, whether, for all graphs and nodes, a positive answer of the tree-CQ implies a positive answer of the GNN. Indeed, we can run the GNN on the body of the tree-CQ (as the graph) and see the result for the free variable: monotonicity under homomorphisms ensures the result.

Second, we can restrict ourselves to tree-CQs of the depth at most the number of layers of the GNN, and there are only a finite number of non-equivalent such tree-CQs. Indeed, the bounded depth and tree shapes ensure this.

Third, for every graph G and node v for which the GNN outputs 1 for (G, v), there is a sound tree-CQ that also outputs 1 for (G, v). Indeed, we can run the GNN on G and convert the "trace" to a tree-CQ.

So, the union of all (non-equivalent) tree-CQs sound for the GNN is what we need.

4.4 Expressivity via Logic: What Else?

Monotonic max GNNs are a trainable neural architecture that shows reasonable performance on many practical tasks [18]. However, it is clearly a rather weak class, and it is a reasonable to ask whether the result of Theorem 19 can be extended to more expressive GNN classes and logics. It is also interesting even if we can guarantee only one direction, the one that some logic is more expressive than some class of GNNs. Indeed, a translation of GNNs to logic is useful, as the result can serve as an explanation, be used to reason together with other logical knowledge, etc. Next we will present three such results, but without the details of the (more complicated) proofs.

First, we consider *monotonic sum GNN* node classifiers, which are the same as monotonic max, except the aggregation function; in particular, they instantiate equation (1) as

$$\boldsymbol{x}_v^{(\ell)} := \sigma\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right),$$

where the other properties and requirements are the same as those for monotonic max GNNs in Definition 21.

For this class, we have the following theorem, which is also due to Tena Cucala et al. [18].

▶ **Theorem 24.** *Monotonic sum GNN node classifiers have less expressive power than unary tree-UCQs with sibling inequalities.*

The logic in this theorem is defined in the same way as normal unary tree-UCQs, except that it relies on *tree-CQs with sibling inequalities* which are the same as tree-CQs except that they additionally allow inequalities between the tree siblings. This logic no longer has a nice grammar; it is not even subsumed by GML, since sibling inequalities give us more than just counting.

Note that Theorem 24 claims an analogue of only one direction of Theorem 19 – that is, that for every monotonic sum GNN node classifier there is a unary tree-UCQ with sibling inequalities realising the same embedding. It is not difficult to show that the other direction does not hold: there are some sibling inequalities not expressible by monotonic sum GNNs. However, even the claimed direction is a much more complicated result than for the max case. It is also much less intuitive, since the result of sum aggregation may grow bigger and bigger when we aggregate with more and more neighbours, and hence a fixed number of

inequalities has to take care of all these arbitrarily large numbers. The key observation for the proof is that, due to monotonicity, if we pass a threshold that is sufficient for a positive overall answer, then we cannot drop below it by adding more neighbours.

Second, we have a look at a recent result similar to the one in Theorem 24, but for a much more expressive class of GNNs and logic. In particular, Benedikt et al. [4] considered *sum-plus GNNs with eventually constant activation*, which instantiate equation (1) as

$$\boldsymbol{x}_v^{(\ell)} := \sigma\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right),$$

where all the $\boldsymbol{A}^{(\ell)}$, $\boldsymbol{C}^{(\ell)}$, $\boldsymbol{b}^{(\ell)}$ are arbitrary matrices and vectors of appropriate dimensions, and the non-linear function σ has the bounds

- t_{left}, such that $\sigma(x) = \sigma(t_{\text{left}})$ for each $x < t_{\text{left}}$, and
- t_{right}, such that $\sigma(x) = \sigma(t_{\text{right}})$ for each $x > t_{\text{right}}$.

Note that this GNN class is indeed much more general than monotonic sum GNNs; however, the condition on σ is not as mild as it may appear at first: truncated ReLU and many other functions satisfy it, but sigmoid and standard ReLU do not. Nonetheless, due to Benedikt et al. [4], we have a logic capturing this.

▶ **Theorem 25.** *Sum-plus GNN node classifiers with eventually constant activation have less expressive power than local modal logic with Presburger quantifiers.*

The definition of this logic is quite elaborate, and its full details are beyond the scope of these notes. We just note that it is an extension of GML with more powerful counting; in particular, instead of quantification of the form $\exists^{\geq n} y.\,(E(x,y) \wedge \varphi(y))$, it has the following quantification, with the intuitive semantics:

$$\sum\nolimits_{i=1}^{n} c_i \cdot \#_y\left(E(x,y) \wedge \varphi(y)\right) \;\geq\; d,$$

where n, the c_i, and d are constant natural numbers.

Finally, we mention GNNs with piecewise-linear activation [8]. This is a very general class of AC-GNNs where aggregation functions include all the common ones (sum, max, min, average, etc.) and the combination functions include all linear transformations of the two arguments with a piecewise-linear function on top (ReLU, truncated ReLU, etc.). We omit the details here, which are quite intricate. The same applies to the logic that captures these GNNs, GFO+C. Thus, we state the result (in fact, only a simplification of the result) without proof and refer interested readers to the paper by Grohe [8] for details. We just mention in the passing that GFO+C is within complexity class TC^0 in the descriptive complexity sense, and the result implies that the same applies to the class of GNNs.

▶ **Theorem 26.** *GNN node classifiers with piecewise-linear activation have less expressive power than GFO+C.*

4.5 Beyond Node Classifiers

When comparing to logic, we focused on inputs with 1-0 node labellings (i.e., those that correspond to logical structures) and $\{yes, no\}$ outputs, which is clearly a significant restriction. In this section, we consider the case of arbitrary labellings in input graphs and $\mathbb{R}$ as the output space (i.e., regression). In this case, we can no longer compare to logic, but we can compare different architectures with each other. In particular, we have the following result by Rosenbluth et al. [15].

▶ **Theorem 27.** *The classes of GNNs that have output space* $\mathbb{R}$ *(and with identity output function) and instantiate equation* (1) *as*

$$\boldsymbol{x}_v^{(\ell)} := \mathit{ReLU}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\mathit{agg}_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right),$$

where agg is one of max, sum, and avg, are pairwise incomparable in expressive power.

We again skip the proof, which in this case is of moderate difficulty, and refer the reader to the original paper [15].

5 Non-Uniform Expressive and Approximation Power

As we have seen, AC-GNNs have three conceptual limitations: they cannot look further than their number of layers, they are "blind" for cycles, since they only see unravellings, and they cannot broadcast information globally in the graph. We may ask ourselves the question whether we can restrict the graphs under consideration so that AC-GNNs can express more than they can over all graphs. For some classes, we may even hope to express all functions – that is, to be *universal*. The following is a very simple result of this kind, which follows from the same argument as we used in Theorem 15: we can take injective aggregation and combination functions, and make the final output function do all the work.

▶ **Proposition 28.** *For every d, AC-GNN node classifiers are universal over trees of diameter (i.e., the length of the longest simple path) bounded by d.*

Note that this result applies for all output sets (i.e., ranges of the realised functions). To get more interesting results, we employ the notion of non-uniform expressive power.

▶ **Definition 29.** *A class* $\mathcal{C}_2$ *of node embeddings has at least as* strong non-uniform expressive power *as class* $\mathcal{C}_1$ *if, for every* $n \in \mathbb{N}$ *and every* $\xi_1 \in \mathcal{C}_1$*, there is* $\xi_2 \in \mathcal{C}_2$ *such that the embeddings of* ξ_1 *and* ξ_2 *agree on all nodes of all graphs with at most n nodes. Classes* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *have the* same non-uniform expressive power *if* $\mathcal{C}_1$ *is also at least as strong as* $\mathcal{C}_2$*.*

Essentially, this notion is "between" distinguishability, which allows its own "simulator" ξ_2 for every two inputs of ξ_1, and uniform expressibility, which requires one single "simulator" for all the inputs. It can also be seen as a restriction on graphs that overcomes the first limitation above. It may also make sense from a practical point of view: we may only be interested in graphs of bounded size.

For this notion, we can start with another simple result. Recall the AC-GNNs with the node update given in equation (3), which we repeat here for convenience:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} \;\Big\|\; \boldsymbol{C}^{(\ell)}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} f(\boldsymbol{x}_u^{(\ell-1)})\right)\right),$$

where f is a smart function such that $\sum_{x\in X} f(x)$ is injective for bounded X (i.e., in our context, for graphs of bounded size). We can make use of the class of all such AC-GNNs (with arbitrary output functions), which we call InAC-GNNs, in the following result by Loukas [10].

▶ **Theorem 30.** *InAC-GNN node classifiers are non-uniform universal over connected graphs with unique node labels.*

Indeed, injective aggregation and combination functions allow us to unambiguously reconstruct the whole graph, and so the output function can do any job we need.

Unique node labels are a rather strict requirement. However, we can relax this and instead require uniqueness only with high probability. This can be achieved by *random node initialisation (RNI)* – that is, where one component of the input graph node labels is drawn from a random distribution. This model is not covered in our formalisation (which does not allow for random inputs and outputs), but one can relativise our setting appropriately, including a definition of expressive power with "high probability." We omit this formalisation and only state the corresponding relativisation of Theorem 30 by Abboud et al. [1].

▶ **Theorem 31.** *InAC-GNN node classifiers with RNI are non-uniform universal over connected graphs.*

To conclude our discussion about non-uniform expressivity, we mention that there is a stronger and deeper result: there is a logic-based formalism that has the same non-uniform expressive power as a very large class of AC-GNNs, which includes nearly all AC-GNNs (with 1-0 input and $\{yes, no\}$ output) that are intuitively "neural" [8]. We again leave this result out of the scope of these notes.

Finally, we briefly introduce a version of approximation power, which has been considered in the context of GNNs.

▶ **Definition 32.** *A class $\mathcal{C}_2$ of node embeddings has at least as* strong (non-uniform) approximation power *as class $\mathcal{C}_1$, if, for every $n \in \mathbb{N}$, every $\epsilon > 0$, and every $\xi_1 \in C_1$, there is $\xi_2^\epsilon \in C_2$ such that embeddings of ξ_1 and ξ_2^ϵ ϵ-agree (i.e., differ by at most ϵ at every position) on all nodes of all graphs with at most n nodes. Classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the* same approximation power *if both directions hold.*

Observe that this definition makes sense only for numeric output spaces, such as regression. One notable result in this category is due to Geerts [6].

▶ **Theorem 33.** *On every compact set of graphs (e.g., where initial labellings are bounded), sum-plus GNNs can approximate any continuous counting bisimulation invariant function (i.e., WL-bounded in terms of distinguishability).*

6 Conclusion

In these notes, we primarily focused on AC-GNNs. However, their limitations in expressivity motivated researchers to invent extensions that allow them to overcome these limitations. For example, they considered the following architectures:

- *recurrent GNNs* [13, 2] have the same aggregation and combination functions for all layers, and iterate them for more than constantly bounded number of times; this allows them to express reachability;
- *k-WL GNNs* [11] consider k-tuples of nodes (pairs, triples, etc.) instead of just nodes; this is the main reason why they can express properties such as the presence of a k-cycle;
- *GNNs with global readouts* [3] allow node labellings to be updated, on each layer, taking into account the aggregation value of all the nodes in the graph; hence, they can express (some) non-local functions – that is, those that depend on nodes with certain properties anywhere in the graph.

All these generalisations (as well as some restrictions, such as GCNs) require their own studies of expressibilty of every kind. Some results already exist, but others are open – that is, the expressibility landscape of GNN-related formalisms is far from being complete for now. There are also many other open problems related to GNNs. For example, it could be practically relevant to design algorithms that produce a reasonably constrained formula (by size, shape, logical language) that is as close as possible, under some notion of proximity, to a given GNN in some GNN language. The result can be seen as a declarative transparent version of the GNN cleared from noise. As a second example, we observe that the fact that some GNN exists does not mean at all that it is possible to train it with any reasonable training procedure from examples, and therefore it could be interesting and practically relevant to define and study "trainable" GNN subclasses.

I hope that these notes could convince you that GNNs and, in particular, their connection to logic and other symbolic formalisms, are a fascinating, but hard topic that not only gives us a lot of insights on the foundations of Machine Learning, but also deserves further studies for the future of AI and CS in general.

References

1 Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *International Joint Conference on Artificial Intelligence*, volume 30, pages 2112–2118, 2021. doi:10.24963/IJCAI.2021/291.

2 Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical characterizations of recurrent graph neural networks with reals and floats. In *Advances in Neural Information Processing Systems*, 2024.

3 Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, volume 8, 2020.

4 Michael Benedikt, Chia-Hsuan Lu, Boris Motik, and Tony Tan. Decidability of graph neural networks via logical characterizations. In *International Colloquium on Automata, Languages, and Programming*, volume 297 of *LIPIcs*, pages 127:1–127:20, 2024. doi:10.4230/LIPICS.ICALP.2024.127.

5 Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

6 Floris Geerts and Juan Reutter. Expressiveness and approximation properties of graph neural networks. In *International Conference on Learning Representations*, volume 10, 2022.

7 Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272, 2017. URL: http://proceedings.mlr.press/v70/gilmer17a.html.

8 Martin Grohe. The descriptive complexity of graph neural networks. In *ACM/IEEE Symposium on Logic in Computer Science*, volume 38, pages 1–14. ACM, 2023. doi:10.1109/LICS56636.2023.10175735.

9 William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30*, pages 1024–1034, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

10 Andreas Loukas. What graph neural networks cannot learn: Depth vs width. In *International Conference on Learning Representations*, volume 8, 2020.

11 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. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019. doi:10.1609/AAAI.V33I01.33014602.

12 Martin Otto. Graded modal logic and counting bisimulation. *arXiv preprint arXiv:1910.00039*, 2019. arXiv:1910.00039.

13 Maximilian Pflüger, David Tena Cucala, and Egor V. Kostylev. Recurrent graph neural networks and their connections to bisimulation and logic. In *AAAI Conference on Artificial Intelligence*, volume 38, 2024.

14 Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, et al. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1):93, 2022.

15 Eran Rosenbluth, Jan Tönshoff, and Martin Grohe. Some might say all you need is sum. In *International Joint Conference on Artificial Intelligence*, volume 32, pages 4172–4179. International Joint Conferences on Artificial Intelligence Organization, 2023. doi:10.24963/IJCAI.2023/464.

16 Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, pages 61–80, 2008.

17 Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, volume 15, pages 593–607. Springer, 2018.

18 David Tena Cucala, Bernardo Cuenca Grau, Boris Motik, and Egor V. Kostylev. On the correspondence between monotonic max-sum gnns and datalog. In *International Conference on Principles of Knowledge Representation and Reasoning*, volume 20, 2023.

19 Boris Weisfeiler and Andrei Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968. Translated from Russian.

20 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, volume 7, 2019.

21 Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *ACM/SIGKDD International Conference on Knowledge Discovery & Data Mining*, volume 24, pages 974–983. ACM, 2018. doi:10.1145/3219819.3219890.

22 Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html.