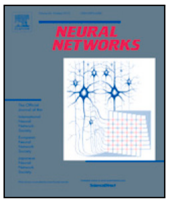
Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Generalization limits of Graph Neural Networks in identity effects learning

Giuseppe Alessio D'Inverno [a,*], Simone Brugiapaglia [b], Mirco Ravanelli [c,d]

[a] *DIISM - University of Siena, via Roma 56, Siena, 53100, Italy*
[b] *Department of Mathematics and Statistics, Concordia University, 1400 De Maisonneuve Blvd. W., Montréal, H3G 1M8, QC, Canada*
[c] *Department of Computer Science and Software Engineering, Concordia University, 2155 Guy St., Montréal, H3H 2L9, QC, Canada*
[d] *Mila - Quebec AI Institute, 6666 Saint-Urbain R., Montréal, H2S 3H1, QC, Canada*

ARTICLE INFO

Keywords:
Graph Neural Networks
Identity effects
Generalization
Encodings
Dicyclic graphs
Gradient descent

ABSTRACT

Graph Neural Networks (GNNs) have emerged as a powerful tool for data-driven learning on various graph domains. They are usually based on a message-passing mechanism and have gained increasing popularity for their intuitive formulation, which is closely linked to the Weisfeiler–Lehman (WL) test for graph isomorphism to which they have been proven equivalent in terms of expressive power. In this work, we establish new generalization properties and fundamental limits of GNNs in the context of learning so-called identity effects, i.e., the task of determining whether an object is composed of two identical components or not. Our study is motivated by the need to understand the capabilities of GNNs when performing simple cognitive tasks, with potential applications in computational linguistics and chemistry. We analyze two case studies: (i) two-letters words, for which we show that GNNs trained via stochastic gradient descent are unable to generalize to unseen letters when utilizing orthogonal encodings like one-hot representations; (ii) dicyclic graphs, i.e., graphs composed of two cycles, for which we present positive existence results leveraging the connection between GNNs and the WL test. Our theoretical analysis is supported by an extensive numerical study.

1. Introduction

Graph Neural Networks (GNNs) (Scarselli, Gori, Tsoi, Hagenbuchner, & Monfardini, 2009) have emerged as prominent models for handling structured data, quickly becoming dominant in data-driven learning over several scenarios such as network analysis (Fan et al., 2020), molecule prediction (Wieder et al., 2020) and generation (Bongini, Bianchini, & Scarselli, 2021), text classification (Malekzadeh et al., 2021), and traffic forecasting (Jiang & Luo, 2022). From the appearance of the earliest GNN model (Scarselli et al., 2009), many variants have been developed to improve their prediction accuracy and generalization power. Notable examples include GraphSage (Hamilton, Ying, & Leskovec, 2017), Graph Attention Networks (Veličković, Cucurull, Casanova, Romero, Lio, & Bengio, 2017), Graph Convolutional Networks (GCN) (Kipf & Welling, 2016), Graph Isomorphism Networks (Xu, Hu, Leskovec, & Jegelka, 2018), and Graph Neural Diffusion (GRAND) (Chamberlain et al., 2021). Furthermore, as the original model was designed specifically for labeled undirected graphs (Scarselli et al., 2009), more complex neural architectures have been designed to handle different types of graph structures, such as directed graphs (Shi, Zhang, Cheng, & Lu, 2019), temporal graphs (Longa et al., 2023), and hypergraphs (Zhang, Zou, & Ma, 2019). For a comprehensive review see, e.g., Hamilton (2020). Over the last decade, there has been growing attention in the theoretical analysis of GNNs. While approximation properties have been examined in different flavors (Azizian & Lelarge, 2020; D'Inverno, Bianchini, Sampoli, & Scarselli, 2024; Keriven & Peyré, 2019; Scarselli et al., 2009), most of the theoretical works in the literature have focused on the *expressive power* of GNNs. From this perspective, the pioneering work of Morris et al. (2019) and Xu et al. (2018) laid the foundation for the standard analysis of GNN expressivity, linking the message-passing iterative algorithm (common to most GNN architectures) to the *first order Weisfeiler Lehman (1–WL) test* (Leman & Weisfeiler, 1968), a popular coloring algorithm used to determine if two graphs are (possibly) isomorphic or not. Since then, the expressive power of GNNs has been evaluated with respect to the 1–WL test or its higher-order variants (called *k*-*-WL test*) (Morris et al., 2019), as well as other variants suited to detect particular substructures (Bodnar, Frasca, Otter et al., 2021; Bodnar, Frasca, Wang et al., 2021). The assessment of the *generalization capabilities* of neural networks has always been crucial for the development of efficient learning algorithms. Several complexity measures have been proposed over

* Corresponding author.
E-mail addresses: dinverno@diism.unisi.it (G.A. D'Inverno), simone.brugiapaglia@concordia.ca (S. Brugiapaglia), mirco.ravanelli@gmail.com (M. Ravanelli).
URLs: https://www3.diism.unisi.it/~dinverno/ (G.A. D'Inverno), http://www.simonebrugiapaglia.ca/ (S. Brugiapaglia),
https://sites.google.com/site/mircoravanelli/ (M. Ravanelli).

https://doi.org/10.1016/j.neunet.2024.106793

Available online 10 October 2024

the past few decades to establish reliable generalization bounds, such as the Vapnik–Chervonenkis (VC) dimension (Vapnik, Levin, & Le Cun, 1994), Rademacher complexity (Bartlett & Mendelson, 2002; Golowich, Rakhlin, & Shamir, 2018), and Betti numbers (Bianchini & Scarselli, 2014). The generalization properties of GNNs have been investigated using these measures. In Scarselli, Tsoi, and Hagenbuchner (2018) the VC dimension of the original GNN model was established, and later extended to message passing-based GNNs by Morris, Geerts, Tönshoff, and Grohe (2023) for piecewise polynomial activation functions. Other generalization bounds for GNNs were derived using the Rademacher complexity (Garg, Jegelka, & Jaakkola, 2020), through a Probably Approximately Correct (PAC) Bayesian approach (Liao, Urtasun, & Zemel, 2020) or using random sampling on the graph nodes (Maskey, Levie, Lee, & Kutyniok, 2022).

An alternative approach for assessing the generalization capabilities of neural networks is based on investigating their ability to learn specific *cognitive tasks* (Marcus, 2003; Marcus, Vijayan, Bandi Rao, & Vishton, 1999; Suárez, Richards, Lajoie, & Misic, 2021), which have long been of primary interest as neural networks were originally designed to emulate functional brain activities. Among the various cognitive tasks, the linguistics community has shown particular interest in investigating so-called *identity effects*, i.e., the task of determining whether objects are formed by two identical components or not (Benua, 1995; Gallagher, 2013). To provide a simple and illustrative example, we can consider an experiment in which the words $\mathsf{AA}, \mathsf{BB}, \mathsf{CC}$ are assigned to the label "good", while $\mathsf{AB}, \mathsf{BC}, \mathsf{AC}$ are labeled as "bad". Now, imagine a scenario where a subject is presented with new test words, such as XX or XY. Thanks to the human ability of abstraction, the subject will be immediately able to classify the new words correctly, even though the letters X and Y were not part of the training set. Identity effects learning finds other examples in, for instance, *reduplication* (which happens when words are inflected by repeating all or a portion of the word) (Paschen, 2021) or *contrastive reduplication* (Ghomeshi, Jackendoff, Rosen, & Russell, 2004). Besides their relevance in linguistics, the analysis of identity effects can serve as an intuitive and effective tool to evaluate the generalization capabilities of neural networks in a variety of specific tasks. These tasks encompass the identification of equal patterns in natural language processing (Wu & Wang, 2010) as well as molecule classification or regression (Wieder et al., 2020). In the context of molecule analysis, the exploitation of molecular symmetries as in, for instance, the class of *bicyclic compounds) (Liebman & Greenberg, 1976)* plays a crucial role as it can be exploited to retrieve molecular orientations (Bunker & Jensen, 2006) or to determine properties of molecular positioning (Pettinari & Santini, 2017). Furthermore, the existence of different symmetries in interacting molecules can lead to different reactions.

Recently, it has been shown in Brugiapaglia, Liu, and Tupper (2022) that Multilayer Perceptrons (MLPs) and Recurrent Neural Networks (RNNs) cannot learn identity effects via Stochastic Gradient Descent nor Adam, under certain conditions on the encoding utilized to represent the components of objects. This finding, based on a framework introduced in Tupper and Shahriari (2016), raises a fundamental question that forms the core focus of our paper: "*Do GNNs possess the capability to learn identity effects?*" Motivated by this research question, this work investigates the generalization limits and capabilities of GNNs when learning identity effects. Our contributions are the following:

(i) extending the analysis of Brugiapaglia et al. (2022), GNNs are shown to be *incapable* of learning identity effects via SGD training under sufficient conditions determined by the existence of a suitable transformation τ of the input space (Theorem 3.1); an application to the problem of classifying identical two-letter words is provided by Theorem 3.3 and supported by numerical experiments in Section 4.2;

(ii) on the other hand, GNNs are shown to be *capable* of learning identity effects in terms of binary classification of *dicyclic graphs*, i.e., graphs composed by two cycles of different or equal length, in Corollary 3.6; a numerical investigation of the gap between our theoretical results and practical performance of GNNs is provided in Section 4.3.

The paper is structured as follows. Section 2 begins by providing a brief overview of fundamental graph theory notation. We then introduce the specific GNN formulation we focus on in our analysis, namely the Weisfeiler–Lehman test, and revisit the framework of rating impossibility theorems for invariant learners. In Section 3, we present and prove our main theoretical results. Section 4 showcases the numerical experiments conducted to validate our findings. Finally, in Section 5, we provide concluding remarks and outline potential avenues for future research.

2. Notation and background

We start by introducing the notation and background concepts that will be used throughout the paper.

2.1. Graph theory basics

A node-attributed graph G is an object defined by a triplet $G = (V, E, \alpha)$. V is the set of *nodes* or *vertices* v, where v can be identified as an element of $\mathbb{N} := \{0, 1, 2, \dots\}$. E is the set of edges $e_{u,v}$, where $e_{u,v} = (u, v) \in V \times V$. The term $\alpha : V \to \mathbb{R}^k$ is the function assigning a *node feature* (or *vertex feature*) α_v to every node v in the graph, with k being the feature dimension. The *number of nodes* of a graph G is denoted by $N := |V|$. All node features can be stacked in a *feature matrix* $\mathbf{X}_G \in \mathbb{R}^{N\times k}$. The *adjacency matrix* $\mathbf{A}$ is defined as $A_{ij} = 1$ if $e_{ij} \in E$, $A_{ij} = 0$ otherwise. The *neighborhood* of a node v is denoted by $\mathcal{N}_v = \{u \mid e_{u,v} \in E\}$.

2.2. Graph neural networks

Graph Neural Networks (GNNs) are a class of connectionist models that aim to learn functions on graphs, or pairs graph/node. Intuitively, a GNN learns how to represent the nodes of a graph by vectorial representations (which are called *hidden states*), giving an encoding of the information stored in the graph. In its general form Gilmer, Schoenholz, Riley, Vinyals, and Dahl (2017) and Scarselli et al. (2009), for each graph $G = (V, E, \alpha) \in \mathcal{G}$ where $\mathcal{G}$ is a node-attributed graph domain, a GNN is defined by the following recursive *updating scheme*:

$$h_v^{(t+1)} = \mathrm{UPDATE}^{(t+1)}\big(h_v^{(t)}, \mathrm{AGGREGATE}^{(t+1)}(\{\!\{ h_u^{(t)} | u \in \mathcal{N}_v \}\!\})\big), \tag{1}$$

for all $v \in V$ and $t = 1, \dots, T$, where $h_v^{(t)}$ is the hidden feature of node v at time t, T is the number of layers of the GNN and $\{\!\{\cdot\}\!\}$ denotes a multiset. Here $\{\mathrm{UPDATE}^{(t)}\}_{t=1,\dots,T}$ and $\{\mathrm{AGGREGATE}^{(t)}\}_{t=1,\dots,T}$ are families of functions that can be defined by learnable or non-learnable schemes. Popular GNN models like GraphSAGE (Hamilton et al., 2017), GCN (Kipf & Welling, 2016), Graph Isomorphism Networks (Xu et al., 2018) are based on this updating scheme. The model terminates with a READOUT function, chosen according to the nature of the task; for instance, global average, min or sum pooling, followed by a trainable multilayer perceptron are typical choices in the case of graph-focused tasks. At a high level, we can formalizate a GNN as a function $g : \mathcal{G} \to \mathbb{R}^r$, where $\mathcal{G}$ is a set of node-attributed graphs and r is the dimension of the output, which depends on the type of task at hand. The updating scheme we choose as a reference for our analysis follows Morris et al. (2019). This model has been proven to match the expressive power of the Weisfeiler–Lehman test (Morris et al., 2019) (see also Theorem 2.1 below), and can therefore be considered a good representative model of the message passing GNN class. The hidden feature $h_v^{(t+1)} \in \mathbb{R}^h$ of a node v at the message passing iteration $t+1$, for $t = 1, \dots, T-1$, is defined as

$$h_v^{(t+1)} = \sigma\big(W_{\text{upd}}^{(t+1)} h_v^{(t)} + W_{\text{agg}}^{(t+1)} h_{\mathcal{N}_v}^{(t)} + b^{(t+1)}\big), \tag{2}$$

where $h_{\mathcal{N}_v}^{(t)} = \text{POOL}\{\!\{h_u^{(t)} | u \in \mathcal{N}_v\}\!\}$, $\sigma : \mathbb{R}^h \to \mathbb{R}^h$ is an element-wise activation function and POOL is the aggregating operator on the neighbor node's features. The aggregating operator can be defined as a non-learnable function, such as the sum, the mean or the minimum, across the hidden features of the neighbors. With respect to Eq. (1), we have that $\text{AGGREGATE}^{(t)}(\cdot) = \text{POOL}(\cdot)\ \forall t = 1, \dots, T$, while $\text{UPDATE}^{(t+1)}(h_v, h_{\mathcal{N}_v}) = \sigma\big(W_{\text{upd}}^{(t+1)} h_v + W_{\text{agg}}^{(t+1)} h_{\mathcal{N}_v} + b^{(t+1)}\big)$. For each node, the initial hidden state is initialized as $h_v^{(0)} = \alpha_v \in \mathbb{R}^k$. The learnable parameters of the GNN can be summarized as $\Theta := (W_{\text{upd}}^{(0)}, W_{\text{agg}}^{(0)}, b^{(0)}, W_{\text{upd}}^{(1)}, W_{\text{agg}}^{(1)}, b^{(1)}, \dots, W_{\text{upd}}^{(L)}, W_{\text{agg}}^{(L)}, b^{(L)})$, with $W_{\text{upd}}^{(0)}, W_{\text{agg}}^{(0)} \in \mathbb{R}^{k\times h}$, $W_{\text{upd}}^{(t)}, W_{\text{agg}}^{(t)} \in \mathbb{R}^{h\times h}$, for $t = 1, \dots, T$, and $b^{(t)} \in \mathbb{R}^h$, for $t = 0, \dots, T$.

2.3. The Weisfeiler–Lehman test

The *first order Weisfeiler–Lehman test* (in short, *1–WL test*) (Leman & Weisfeiler, 1968) is one of the most popular isomorphism tests for graphs, based on an iterative coloring scheme. The coloring algorithm is applied in parallel to two input graphs, giving a color partition of the nodes as output. If the partitions match, then the graphs are possibly isomorphic, while if they do not match, then the graphs are certainly non-isomorphic. Note that the test is not conclusive in the case of a positive answer, as the graphs may still be non-isomorphic; nevertheless, the 1–WL test provides an accurate isomorphism test for a large class of graphs (Babai & Kucera, 1979). The coloring is carried out by an iterative algorithm which takes as input a graph $G = (V, E, \alpha)$ and, at each iteration, computes a *node coloring* $c^{(t)}(v) \in \mathcal{C}$ for each node $v \in V$, being $\mathcal{C} \subseteq \mathbb{N}$ a subset of natural numbers representing colors. The algorithm is sketched in the following.

1. At iteration 0, in the case of labeled graphs, the node color initialization is based on the vertex feature according to a specific hash function $\text{HASH}_0 : \mathbb{R}^k \to \mathcal{C}$; namely, $c^{(0)}(v) = \text{HASH}_0(\alpha(v))$, for all $v \in V$. For unlabeled graphs, a node color initialization is provided, usually setting every color as equal to a given initial color $c^{(0)} \in \mathcal{C}$.
2. For any iteration $t > 0$, we set

   $$c^{(t)}(v) = \text{HASH}((c^{(t-1)}(v), \{\!\{c^{(t-1)}(n) | n \in \mathcal{N}_v\}\!\})),$$

   $\forall v \in V$, where HASH injectively maps the above color-multiset pair to a unique value in $\mathcal{C}$.

The algorithm terminates if the number of colors between two iterations does not change, i.e., when the cardinalities of $\{c^{(t-1)}(v) | v \in V\}$ and $\{c^{(t)}(v) | v \in V\}$, namely, are equal.

We conclude by recalling two results establishing the equivalence between GNNs' and 1–WL test's expressive power that will be instrumental for our analysis. A first result was proved in Xu et al. (2018) and it characterizes the equivalence on a graph-level task for GNNs with generic message passing layers satisfying suitable conditions.

Another characterization, reported below, is due to Morris et al. (2019) and states the equivalence on a node coloring level, referring to the particular model defined in (2).

Theorem 2.1 (*See Morris et al. (2019, Theorem 2)*). *Let $G = (V, E, \alpha)$ be a graph with initial coloring $c^{(0)}(v) \in \mathbb{R}$ for each node $v \in V$ (so that $c^{(0)} \in \mathbb{R}^{|V(G)|}$). Then, for all $t \geq 0$ there exists a GNN of the form (2) such that the hidden feature vector $h^{(t)} \in \mathbb{R}^{|V(G)|}$ produced by the GNN at layer t coincides with the color vector $c^{(t)} \in \mathbb{R}^{|V(G)|}$ produced by the 1–WL test at iteration t, i.e., $c^{(t)} \equiv h^{(t)}$.*

2.4. Rating impossibility for invariant learners

We now recall the framework of rating impossibility from Brugiapaglia et al. (2022), which we will then apply to the case of identity effects learning. In general, we assume to train a *learning algorithm* to perform a rating assignment task, where the rating r is a real number. Let $\mathcal{I}$ be the set of all possible inputs x (that could be, for instance, elements of $\mathbb{R}^d$). Our learning algorithm is trained on a dataset $D \subseteq \mathcal{I} \times \mathbb{R}$ consisting of a finite set of input-rating pairs (x, r). Let $\mathcal{D}$ be the set of all possible datasets with inputs in $\mathcal{I}$. The learning algorithm is trained via a suitable optimization method, such as Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (Adam) (Kingma & Ba, 2014), that for any given training dataset D outputs the optimized set of parameters $\Theta = \Theta(D) \in \mathbb{R}^p$, which, in turn, defines a model $f = f(\Theta, \cdot)$. The rating prediction on a novel input $x \in \mathcal{I}$ is then given by $r = f(\Theta, x)$. In summary, a learning algorithm can thought of as a map $L : \mathcal{D} \times \mathcal{I} \to \mathbb{R}$, defined as $L(D, x) = f(\Theta(D), x)$.

Given the stochastic nature of neural network training, we adopt a nondeterministic point of view. Hence we require the notion of *equality in distribution*. Two random variables X, Y taking values in $\mathbb{R}^k$ are said to be *equal in distribution* (denoted by $X \overset{d}{=} Y$) if $\mathbb{P}(X \leq x) = \mathbb{P}(Y \leq x)$ for all $x \in \mathbb{R}^k$, where inequalities hold componentwise.

With this notation, rating impossibility means that $L(D, x_1) \overset{d}{=} L(D, x_2)$ for two inputs $x_1 \neq x_2$ drawn from $\mathcal{I} \setminus D$. Sufficient conditions for rating impossibility are identified by the following theorem from Brugiapaglia et al. (2022) (here slightly adapted using equality in distribution), which involves the existence of an auxiliary transformation τ of the inputs.

Theorem 2.2 (*Rating Impossibility for Invariant Learners, Brugiapaglia et al. (2022, Theorem 1)*). *Consider a dataset $D \subseteq \mathcal{I} \times \mathbb{R}$ and a transformation $\tau : \mathcal{I} \to \mathcal{I}$ such that*

(i) $\tau(D) \overset{d}{=} D$ (invariance of the data).[1]

Then, for any learning algorithm $L : \mathcal{D} \times \mathcal{I} \to \mathbb{R}$ and any input $x \in \mathcal{I}$ such that

(ii) $L(\tau(D), \tau(x)) \overset{d}{=} L(D, x)$ (invariance of the algorithm),

we have $L(D, \tau(x)) \overset{d}{=} L(D, x)$.

This theorem states that under the invariance of the data and of the algorithm, the learner cannot assign different ratings to an input x and its transformed version $\tau(x)$. This leads to rating impossibility when $\tau(x) \neq x$ and $x, \tau(x) \in \mathcal{I} \setminus D$.

We conclude by recalling some basic notions on SGD training. Given a dataset D, we aim to find parameters Θ that minimize an objective function of the form

$$F(\Theta) = \mathcal{L}((f(\Theta, x), r) : (x, r) \in D), \quad \Theta \in \mathbb{R}^p,$$

where $\mathcal{L}$ is a (possibly regularized) loss function. We assume F to be differentiable over $\mathbb{R}^p$ in order for its gradients to be well defined. Given a collection of subsets $(D_i)_{i=0}^{k-1}$ with $D_i \subseteq D$ (usually referred to as training batches, which can be either deterministically or randomly generated), we define F_{D_i} as the function F where the loss is evaluated only on data in D_i. In SGD-based training, we randomly initialize Θ_0 and iteratively compute

$$\Theta_{i+1} = \Theta_i - \eta_i \frac{\partial F_{D_i}}{\partial \Theta}(\Theta_i), \tag{3}$$

for $i = 0, 1, \dots, k-1$, where the sequence of step sizes $(\eta_i)_{i=0}^{k-1}$ is assumed to be either deterministic or random and independent of $(D_i)_{i=0}^{k-1}$. Note that, being Θ_i a random vector for each i, the output of the learning algorithm $L(D, x) = f(\Theta_k, x)$ is a random variable.

[1] By definition, $\tau(D) := \{(\tau(x), r) : (x, r) \in D\}$.

3. Theoretical analysis

In this section we present our theoretical analysis. More specifically, in Section 3.1 we establish a rating impossibility theorem for GNNs under certain technical assumptions related to the invariance of the training data under a suitable transformation τ of the inputs; then, we illustrate an application to the case study of identity effects learning for a two-letter word dataset in Section 3.1.1. In Section 3.2 we prove that symmetric dicyclic graphs can be distinguished from the asymmetric ones by the 1–WL test, and consequently by a GNN.

3.1. What GNNs cannot learn: rating impossibility theorem

We assume the input space to be of the form $\mathcal{I} = \mathbb{R}^d \times \mathbb{R}^d$ and the learning algorithm

$$L(D, x) = f(B, Gu + Hv, Hu + Gv), \ \forall x = (u, v) \in \mathcal{I}, \tag{4}$$

where $\Theta = (B, G, H)$ are trainable parameters and $G, H \in \mathbb{R}^{d\times d}$. This class of learning algorithms perfectly fits the formulation given in Morris et al. (2019), where the updating scheme is the one defined by (2). In this case,

$$G = W^{(1)}_{upd}, \quad H = W^{(1)}_{agg},$$
$$B = \left(b^{(1)}, W^{(2)}_{upd}, W^{(2)}_{agg}, b^{(2)} \ldots, W^{(N)}_{upd}, W^{(N)}_{agg}, b^{(N)}\right).$$

The learner defined by Eq. (4) mimics, in this specific setting, the behavior of several GNN architectures, GCN included. In fact, when the graph is composed by only two nodes, the convolution ends up being a weighted sum of the hidden states of the two nodes, i.e., $h^{(t)}_{\mathcal{N}_t} = h^{(t)}_u$ and

$$h^{(t+1)}_v = \sigma\left(W^{(t+1)}_{\text{upd}} h^{(t)}_v + W^{(t+1)}_{\text{agg}} h^{(t)}_u + b^{(t+1)}\right).$$

This property will have practical relevance in Theorem 3.3 and its experimental realization in Section 4.2.

In the following result we identify sufficient conditions on the dataset D and the training procedure able to guarantee invariance of GNN-type models of the form (4) trained via SGD to a suitable class of transformations τ (hence verifying condition (ii) of Theorem 2.2).

Theorem 3.1 (*Invariance of GNN-Type Models Trained Via SGD*). *Assume the input space to be of the form $\mathcal{I} = \mathbb{R}^d \times \mathbb{R}^d$. Let $\tau : \mathcal{I} \to \mathcal{I}$ be a linear transformation defined by $\tau(x) = (u, \tau_2(v))$ for any $x = (u, v) \in \mathcal{I}$, where $\tau_2 : \mathbb{R}^d \to \mathbb{R}^d$ is also linear. Moreover, assume that*

- *the matrix $T_2 \in \mathbb{R}^{d\times d}$ associated with the transformation τ_2 is orthogonal and symmetric;*
- *the dataset $D = \{((u_i, v_i), r_i)\}_{i=1}^n$ is invariant under the transformation $\tau_2 \otimes \tau_2$, i.e.,*

$$(u_i, v_i) = \left(\tau_2(u_i), \tau_2(v_i)\right), \quad \forall i = 1, \ldots, n. \tag{5}$$

Suppose k iterations of SGD as defined in (3) are used to determine parameters $\Theta_k = (B_k, G_k, H_k)$ with objective function

$$F(\Theta) = \sum_{i=1}^n \ell\left(f(B, Gu_i + Hv_i, Hu_i + Gv_i), r_i\right) + \lambda \mathcal{R}(B),$$

for some $\lambda \geq 0$, with $\Theta = (B, G, H)$ and where ℓ, f and $\mathcal{R}$ are real-valued functions such that F is differentiable. Suppose the random initialization of the parameters B, G and H to be independent and that the distributions of G_0 and H_0 are invariant with respect to right-multiplication by T_2. Then, the learner L defined by $L(D, x) = f(B_k, G_k u + H_k v, H_k u + G_k v)$, for $x = (u, v)$, satisfies $L(D, x) \overset{d}{=} L(\tau(D), \tau(x))$.

Proof. Given a batch $D_i \subseteq D$, define $J_i := \{j \in \{1, \ldots, n\} : ((u_j, v_j), r_j) \in D_i\}$ and

$$F_{D_i}(\Theta) = \sum_{j\in J_i} \ell(f(B, Gv_j + Hu_j, Hv_j + Gu_j), r_j) + \lambda \mathcal{R}(B).$$

Moreover, consider an *auxiliary objective function*, defined by

$$\tilde{F}_{D_i}(B, G_1, H_1, H_2, G_2) = \sum_{j\in J_i} \ell(f(B, G_1 v_j + H_1 u_j, H_2 v_j + G_2 u_j), r_j) + \lambda \mathcal{R}(B).$$

Observe that $F_{D_i}(\Theta) = \tilde{F}_{D_i}(B, G, H, H, G)$. Moreover,

$$\frac{\partial F_{D_i}}{\partial B}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial B}(\Theta) \tag{6}$$
$$\frac{\partial F_{D_i}}{\partial G}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial G_1}(\Theta) + \frac{\partial \tilde{F}_{D_i}}{\partial G_2}(\Theta) \tag{7}$$
$$\frac{\partial F_{D_i}}{\partial H}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial H_1}(\Theta) + \frac{\partial \tilde{F}_{D_i}}{\partial H_2}(\Theta) \tag{8}$$

Moreover, replacing D_i with its transformed version $\tau(D_i) = \{((u_j, \tau_2(v_j)), r_j)\}_{j\in D_i}$, we see that $F_{\tau(D_i)}(\Theta) = \tilde{F}_{D_i}(B, G, HT_2, H, GT_2)$. This leads to

$$\frac{\partial F_{\tau(D_i)}}{\partial B}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial B}(B, G, HT_2, H, GT_2) \tag{9}$$
$$\frac{\partial F_{\tau(D_i)}}{\partial G}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial G_1}(B, G, HT_2, H, GT_2) + \frac{\partial \tilde{F}_{D_i}}{\partial G_2}(B, G, HT_2, H, GT_2)T_2^T \tag{10}$$
$$\frac{\partial F_{\tau(D_i)}}{\partial H}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial H_1}(B, G, HT_2, H, GT_2)T_2^T + \frac{\partial \tilde{F}_{D_i}}{\partial H_2}(B, G, HT_2, H, GT_2). \tag{11}$$

Now, denoting $\ell = \ell(f, r)$ and $f = f(B, u, v)$, we have

$$\frac{\partial \tilde{F}_{D_i}}{\partial G_1} = \sum_{j\in D_i} \frac{\partial \ell}{\partial f}\frac{\partial f}{\partial u} v_j^T, \quad \frac{\partial \tilde{F}_{D_i}}{\partial H_1} = \sum_{j\in D_i} \frac{\partial \ell}{\partial f}\frac{\partial f}{\partial u} u_j^T,$$
$$\frac{\partial \tilde{F}_{D_i}}{\partial H_2} = \sum_{j\in D_i} \frac{\partial \ell}{\partial f}\frac{\partial f}{\partial v} v_j^T, \quad \frac{\partial \tilde{F}_{D_i}}{\partial G_2} = \sum_{j\in D_i} \frac{\partial \ell}{\partial f}\frac{\partial f}{\partial v} u_j^T.$$

In addition, thanks to assumption (5), we have $u_j^T T_2^T = u_j^T$ and $v_j^T T_2^T = v_j^T$ for all $j \in J_i$. Thus, we obtain

$$\frac{\partial \tilde{F}_D}{\partial G_1} T_2^T = \frac{\partial \tilde{F}_D}{\partial G_1}, \quad \frac{\partial \tilde{F}_D}{\partial H_1} T_2^T = \frac{\partial \tilde{F}_D}{\partial H_1}, \tag{12}$$
$$\frac{\partial \tilde{F}_D}{\partial H_2} T_2^T = \frac{\partial \tilde{F}_D}{\partial H_2}, \quad \frac{\partial \tilde{F}_D}{\partial G_2} T_2^T = \frac{\partial \tilde{F}_D}{\partial G_2}. \tag{13}$$

Now, let $(B_0', G_0', H_0') \overset{d}{=} (B_0, G_0, H_0)$ and let (B_i', G_i', H_i') for $i = 1, \ldots, k$ be the sequence generated by SGD, applied to the transformed data $\tau(D)$. By assumption, we have $B_0' \overset{d}{=} B_0$, $G_0 \overset{d}{=} G_0' \overset{d}{=} G_0' T_2$ and $H_0 \overset{d}{=} H_0' \overset{d}{=} H_0' T_2$. We now show by induction that $B_i' \overset{d}{=} B_i$, $G_i \overset{d}{=} G_i' \overset{d}{=} G_i' T_2$ and $H_i \overset{d}{=} H_i' \overset{d}{=} H_i' T_2$ for all indices $i = 1, \ldots, k$. Using Eqs. (6) and (9) and the inductive hypothesis, we have

$$\begin{aligned} B_{i+1}' &= B_i' - \eta_i \frac{\partial F_{\tau(D_i)}}{\partial B}(B_i', G_i', H_i') \\ &= B_i' - \eta_i \frac{\partial \tilde{F}_{D_i}}{\partial B}(B_i', G_i', H_i' T_2, H_i', G_i' T_2) \\ &\overset{d}{=} B_i - \eta_i \frac{\partial \tilde{F}_{D_i}}{\partial B}(B_i, G_i, H_i, H_i, G_i) \\ &= B_i - \eta_i \frac{\partial F_{\tau(D_i)}}{\partial B}(B_i, G_i, H_i) = B_{i+1}. \end{aligned}$$

Similarly, using Eqs. (7), (10) and (13) and the inductive hypothesis, we see that

$$G_{i+1}' = G_i' - \eta_i \frac{\partial F_{\tau(D_i)}}{\partial G}(B_i', G_i', H_i')$$

$$\begin{aligned}
&= G_i' - \eta_i \left(\frac{\partial \tilde{F}_{D_i}}{\partial I}(B_i', G_i', H_i'T_2, H_i', G_i'T_2) \right. \\
&\quad \left. + \frac{\partial \tilde{F}_{D_i}}{\partial L}(B_i', G_i', H_i'T_2, H_i', G_i'T_2)T_2^T \right) \\
&= G_i' - \eta_i \left(\frac{\partial \tilde{F}_{D_i}}{\partial G_1}(B_i', G_i', H_i'T_2, H_i', G_i'T_2) \right. \\
&\quad \left. + \frac{\partial \tilde{F}_{D_i}}{\partial G_2}(B_i', G_i', H_i'T_2, H_i', G_i'T_2) \right) \\
&\overset{d}{=} G_i - \eta_i \left(\frac{\partial \tilde{F}_{D_i}}{\partial G_1}(B_i, G_i, H_i, H_i, G_i) \right. \\
&\quad \left. + \frac{\partial \tilde{F}_{D_i}}{\partial G_2}(B_i, G_i, H_i, H_i, G_i) \right) \\
&= G_i - \eta_i \frac{\partial F_{D_i}}{\partial G}(B_i, G_i, H_i) = G_{i+1}.
\end{aligned}$$

One proceeds analogously for H_{i+1}' using Eqs. (8), (11) and (12). Similarly, one also sees that $G_{i+1}'T_2 \overset{d}{=} G_{i+1}$ and $H_{i+1}'T_2 \overset{d}{=} H_{i+1}$ combining the previous equations with symmetry and orthogonality of T_2.

In summary, we have

$$\begin{aligned}
L(D,x) &= f(B_k, G_k u + H_k v, H_k u + G_k v) \\
&\overset{d}{=} f(B_k', G_k' u + H_k' v, H_k' u + G_k' v) \\
&\overset{d}{=} f(B_k', G_k' u + H_k' T_2 v, H_k' u + G_k' T_2 v) \\
&= L(\tau(D), \tau(x)),
\end{aligned}$$

which concludes the proof. □

Remark 3.2 (*On the Assumptions of Theorem 3.1*)**.** At first glance, the assumptions of Theorem 3.1 might seem quite restrictive, especially the assumption about the invariance of the distributions of G_0 and H_0 with respect to right-multiplication by the symmetric orthogonal matrix T_2. Yet, this hypothesis holds, e.g., when the entries of G_0 and H_0 are independently and identically distributed according to a centered normal distribution thanks to the rotational invariance of isotropic random Gaussian vectors (see, e.g., Vershynin (2018, Proposition 3.3.2)). This is the case in common initialization strategies such as Xavier initialization (Glorot & Bengio, 2010). In addition, numerical results presented in Section 4 suggest that rating impossibility might hold in more general settings, such as when the model f includes ReLU activations (hence, when F has points of nondifferentiability) or for models trained via Adam as opposed to SGD.

3.1.1. *Application to identity effects*

As a practical application of Theorem 3.1 to identity effects, we consider the problem of classifying identical two-letter words of the English alphabet $\mathcal{A} := \{\mathsf{A}, \mathsf{B}, \dots, \mathsf{Z}\}$, already mentioned in Section 1 and following Brugiapaglia et al. (2022). Consider a training set D formed by two-letter words that do not contain Y nor Z. Words are assigned the label 1 if they are composed by identical letters and 0 otherwise. Our goal is to verify whether a learning algorithm is capable of generalizing this pattern correctly to words containing the letters Y or Z. The transformation τ of Theorem 3.1 is defined by

$$\tau(\mathsf{xY}) = \mathsf{xZ}, \ \tau(\mathsf{xZ}) = \mathsf{xY}, \text{ and } \tau(\mathsf{xy}) = \mathsf{xy}, \tag{14}$$

for all letters $\mathsf{x}, \mathsf{y} \in \mathcal{A}$, with $\mathsf{y} \neq \mathsf{Y}, \mathsf{Z}$. Note that this transformation is of the form $\tau = I \otimes \tau_2$, where I is the identity map. Hence, it fits the setting of Theorem 3.1. Moreover, since D does not contain Y nor Z letters, $\tau(D) = D$. Hence, condition (i) of Theorem 2.2 is satisfied.

In order to represent letters as vectors of $\mathbb{R}^d$, we need to use a suitable *encoding*. Its choice is crucial to determine the properties of the transformation matrix T_2 associated with τ_2, needed to apply Theorem 3.1. Formally, an encoding of an alphabet $\mathcal{A}$ is a set of vectors $\mathcal{E} \subseteq \mathbb{R}^d$, of the same cardinality of $\mathcal{A}$, to which letters can be associated with. In our case, $|\mathcal{A}| = 26 = |\mathcal{E}|$. We say that an encoding is *orthogonal* if it is an orthonormal set of $\mathbb{R}^d$. For example, the popular one-hot encoding $\mathcal{E} = \{e_i\}_{i=1}^{26} \subseteq \mathbb{R}^{26}$, i.e., the canonical basis of $\mathbb{R}^{26}$, is an orthogonal encoding.

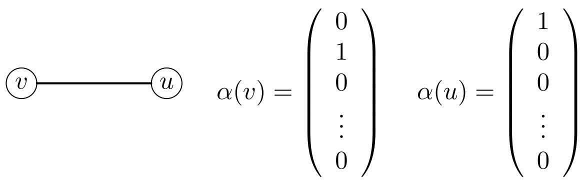

v
u
$\alpha(v) = (0, 1, 0, \vdots, 0)^T$
$\alpha(u) = (1, 0, 0, \vdots, 0)^T$

Fig. 1. Graph modeling of a two-letter word: a vertex feature $\alpha(v) \in \mathbb{R}^d$ is attached to each node v of a two-node undirected graph, according to a given encoding $\mathcal{E}$ of the English alphabet $\mathcal{A}$. In this figure, $\mathcal{E}$ is the one-hot encoding.

In this setting, every word is modeled as a graph defined by two nodes connected by a single unweighted and undirected edge. Each node v is labeled with a node feature $\alpha(v) \in \mathbb{R}^d$, corresponding to a letter's encoding. An example is depicted in Fig. 1.

Theorem 3.3 (*Inability of GNNs to classify identical two-letter words outside the training set*)**.** *Let $\mathcal{E} \subseteq \mathbb{R}^{26}$ be an orthogonal encoding of the English alphabet $\mathcal{A}$ and let L be a learner obtained by training a GNN of the form (2) via SGD to classify identical two-letter words. Assume that words in the training set D do not contain the letter* Y *nor* Z*. Then, L assigns the same rating (in distribution) to any word of the form* xy *where* $\mathsf{y} \in \{\mathsf{Y}, \mathsf{Z}\}$*, i.e., $L(D, \mathsf{xY}) \overset{d}{=} L(D, \mathsf{xZ})$ for any $\mathsf{x} \in \mathcal{A}$. Hence, it is unable to generalize to identity effect outside the training set.*

Proof. As discussed above, the transformation τ defined by (14) is of the form $\tau = I \otimes \tau_2$. Moreover, the matrix associated with the linear transformation τ_2 is of the form $T_2 = B^{-1}PB$, where B is the change-of-basis matrix from the orthonormal basis associated with the encoding $\mathcal{E}$ to the canonical basis of $\mathbb{R}^{26}$ (in particular, B is orthogonal and $B^{-1} = B^T$) and P is a permutation matrix that switches the last two entries of a vector, i.e., using block-matrix notation,

$$P = \left[\begin{array}{c|cc} I & \multicolumn{2}{c}{0} \\ \hline \multirow{2}{*}{0} & 0 & 1 \\ & 1 & 0 \end{array}\right], \quad I \in \mathbb{R}^{24 \times 24}.$$

Hence, T_2 is orthogonal and symmetric, and therefore fits the framework of Theorem 3.1.

On the other hand, as discussed in Section 3.1, every GNN of the form (2) is a model of the form (4). Thus, Theorem 3.1 yields $L(D, \mathsf{xy}) \overset{d}{=} L(\tau(D), \tau(\mathsf{xy}))$, for all letters $\mathsf{x}, \mathsf{y} \in \mathcal{A}$. In particular, $L(D, \mathsf{xY}) \overset{d}{=} L(\tau(D), \mathsf{xZ})$, which corresponds to condition (ii) of Theorem 2.2. Recalling that $\tau(D) = D$, also condition (i) holds. Hence, we can apply Theorem 2.2 and conclude the proof. □

3.2. *What GNNs can learn: identity effects on dicyclic graphs*

We now analyze the expressivity of GNNs to learn identity effects related to the *topology* of the graphs in the dataset. This novel setting requires to design *ex novo* the formulation of our problem. In fact, we are not focusing on the feature matrix X_G of a graph anymore, but on its adjacency matrix A, which contains all the topological information. Here we focus on a particular class of graphs, which we call *dicyclic graphs*. A dicyclic graph is a graph composed by an m-cycle and an n-cycle, linked by a single edge. Since a dicyclic graph is uniquely determined by the length of the two cycles, we can identify it with the equivalence class $[m, n]$ over the set of pairs (a, b), $a, b \in \mathbb{N}$, defined as $[m,n] := \{(m,n),(n,m)\}$. A dicyclic graph $[m,n]$ is *symmetric* if $m = n$ and *asymmetric* otherwise.

In this section we provide an analysis of the expressive power of GNNs when learning identity effects on dicyclic graphs (i.e., classifying whether a dicyclic graph is symmetric or not). We start by proving a lemma that shows how information propagates through the nodes of a cycle, during the 1–WL test iterations, when one of the nodes has a different initial color with respect to all the other nodes.

Lemma 3.4 (*1-WL Test On m-Cycles*). *Consider an m-cycle in which the vertices are numbered from 0 to $m-1$ clockwise, an initial coloring $c^{(0)} = [0,1,\dots,1]^T \in \mathbb{N}^m$ (vector indexing begins from 0, and the vector is meant to be circular, i.e., $c^{(0)}(m) = c^{(0)}(0)$), and define the function* HASH *as*

$$\begin{cases} \text{HASH}(0, \{\!\{j,k\}\!\}) = 0 & \\ \text{HASH}(i, \{\!\{j,k\}\!\}) = i & \text{if } j \neq k,\ i < \lfloor \frac{m}{2} \rfloor \\ \text{HASH}(i, \{\!\{j,k\}\!\}) = i+1 & \text{if } j = k,\ i < \lfloor \frac{m}{2} \rfloor \\ \text{HASH}(\lfloor \frac{m}{2} \rfloor, \{\!\{j,k\}\!\}) = \lfloor \frac{m}{2} \rfloor & \end{cases},$$

with $j,k \le \lfloor \frac{m}{2} \rfloor$. Then, HASH *is an injective coloring over the m-cycle at each iteration t of the 1–WL test. This gives, at each iteration $0 \le t < T = \lfloor \frac{m}{2} \rfloor$, the coloring*

$$\begin{cases} c^{(t)}(i) = i & \text{if } 0 \le i \le t \\ c^{(t)}(i) = t+1 & \text{if } t < i < m-t \\ c^{(t)}(i) = m-i & \text{if } m-t \le i < m \end{cases}, \tag{15}$$

and the 1-WL test terminates after $T = \lfloor \frac{m}{2} \rfloor$ iterations (i.e., $c^{(T)} = c^{(T-1)}$), giving $\lfloor \frac{m}{2} \rfloor + 1$ colors.

Proof. We prove the lemma by induction on t. *Case $t = 1$* We start with $c^{(0)}(0) = 0$ and $c^{(0)}(i) = 1$, for $i = 1,\dots,m-1$. We only have three hashing cases:

- HASH$(0, \{\!\{1,1\}\!\}) = 0$, the color assigned to node 0;
- HASH$(1, \{\!\{0,1\}\!\}) = 1$, the color assigned to nodes 1 and $m-1$;
- HASH$(1, \{\!\{1,1\}\!\}) = 2$, the color assigned to all nodes $1 < i < m-1$.

This shows that $c^{(1)}$ satisfies (15) and that HASH is injective at iteration $t = 1$. Hence, the claim is true for $t = 1$. *Inductive step $t \to t+1$* Assume that the inductive hypothesis is true for step t. Hence, our coloring is of the form (15) and that HASH is injective at iteration t. This means that for $0 < i \le t$ we have $c^{(t)}(i-1) < c^{(t)}(i) < c^{(t)}(i+1)$ and for $m-t \le i < m-1$ we have $c^{(t)}(i+1) < c^{(t)}(i) < c^{(t)}(i-1)$; thus, for $0 < i \le t$ or $m-t-1 \le i < m-1$, we see that

$$c^{(t+1)}(i) = \text{HASH}(c^{(t)}(i), \{\!\{c^{(t)}(i-1), c^{(t)}(i+1)\}\!\}) = i.$$

For $i = t+1$ we have $c^{(t)}(i-1) < c^{(t)}(i) = c^{(t)}(i+1)$ and for $i = m-t-2$ we have $c^{(t)}(i+1) < c^{(t)}(i) = c^{(t)}(i-1)$; therefore, for $i = t+1$ and $i = m-t-2$, we also have

$$c^{(t+1)}(i) = \text{HASH}(c^{(t)}(i), \{\!\{c^{(t)}(i-1), c^{(t)}(i+1)\}\!\}) = i.$$

For all the remaining indices $t+1 < i < m-t-2$, we have $c^{(t)}(i-1) = c^{(t)}(i) = c^{(t)}(i+1)$, so

$$\begin{aligned} c^{(t+1)}(i) &= \text{HASH}(c^{(t)}(i), \{\!\{c^{(t)}(i-1), c^{(t)}(i+1)\}\!\}) \\ &= (t+1)+1 = t+2. \end{aligned}$$

The HASH function is still injective, as for $0 < i \le t+1$ we have $c^{(t)}(i-1) < c^{(t)}(i) < c^{(t)}(i+1)$, for $m-t-1 \le i < m-1$ we have $c^{(t)}(i+1) < c^{(t)}(i) < c^{(t)}(i-1)$, and for $t+1 < i < m-t-1$ it holds HASH$(c^{(t)}(i), \{\!\{c^{(t)}(i-1), c^{(t)}(i+1)\}\!\})$ = HASH$(t+1, \{\!\{t+1, t+1\}\!\}) = t+2$. Therefore, we have

$$\begin{cases} c^{(t+1)}(i) = i & \text{if } 0 \le i \le t+1 \\ c^{(t+1)}(i) = t+2 & \text{if } t+1 < i < m-t-1 \\ c^{(t+1)}(i) = m-i & \text{if } m-t-1 \le i < m \end{cases}.$$

Termination of the 1–WL test
At iteration $\lfloor \frac{m}{2} \rfloor - 1$ we have

$$\begin{cases} c^{(\lfloor \frac{m}{2} \rfloor -1)}(i) = i & \text{if } 0 \le i \le \lfloor \frac{m}{2} \rfloor - 1 \\ c^{(\lfloor \frac{m}{2} \rfloor -1)}(i) = \lfloor \frac{m}{2} \rfloor & \text{if } i = \lfloor \frac{m}{2} \rfloor \text{ or } i = \lceil \frac{m}{2} \rceil \\ c^{(\lfloor \frac{m}{2} \rfloor -1)}(i) = m-i & \text{if } \lceil \frac{m}{2} \rceil + 1 \le i < m \end{cases}.$$

This concludes the proof. □

A graphical representation of Lemma 3.4 can be found in Fig. 2. We observe that the specific node indexing of Lemma 3.4 was adopted just to ease computations; nevertheless, it is possible to construct a HASH function for other choices of node indexing. This is due to the fact that the mapping depends only on the topological structure in each node's neighborhood. This lemma represents the core of next theorem's proof, which establishes the ability of the 1-WL test to classify dicyclic graphs with identical cycles. Intuitively, if we have a dicyclic graph where node colors are uniformly initialized, one step of 1–WL test yields a coloring depending entirely on the number of neighbors for each node. In a dicyclic graph $[m,n]$ we always have $m+n-2$ nodes of degree two and 2 nodes of degree three, so $c^{(1)}(i) = 1$ for all 2-degree nodes i, and $c^{(1)}(j) = 0$ for the two 3-degree nodes j. Hence, each cycle of the dicyclic graph satisfies the initial coloring hypothesis of Lemma 3.4.

Theorem 3.5 (*1-WL Test on Dicyclic Graphs*). *The 1–WL test gives the same color to the 3-degree nodes of a uniformly colored dicyclic graph $[m,n]$ (i.e., $c^{(0)} = 0 \in \mathbb{N}^{m+n}$) if and only if $m = n$. Therefore, the 1–WL test can classify symmetric dicyclic graphs.*

Proof. After one iteration on the 1–WL test, regardless of the symmetry of the dicyclic graph, we obtain a coloring in which only 3-degree nodes have a different color, whose value we set to 0. We can therefore split the coloring vector $c^{(1)} \in \mathbb{N}^{m+n}$ in two subvectors, namely, $c^{(1)} = [(c_1^{(1)})^T, (c_2^{(1)})^T]^T$ corresponding to each cycle, respectively, and where $c_1^{(1)}(0)$ and $c_2^{(1)}(0)$ correspond to the 3-degree nodes. We treat the symmetric and the asymmetric cases separately.

The symmetric case We let $c_1^{(0)} = c_2^{(0)} = c_0^{(0)}$, with $c_0^{(0)} = [0,1,\dots,1]$. In this case, we run the 1–WL test in parallel on both vectors $c_1^{(t)}$ and $c_2^{(t)}$, where the HASH function in Lemma 3.4 is extended on the 3-degree nodes as HASH$(0, \{\!\{0,j,k\}\!\}) = 0$. Therefore, for each $t \ge 0$,

$$c_0^{(t+1)}(0) = \text{HASH}(c_0^{(t)}(0), \{\!\{c_0^{(t)}(0), c_0^{(t)}(1), c_0^{(t)}(m-1)\}\!\}) = 0.$$

Thanks to Lemma 3.4 we obtain $c_1^{(\lfloor \frac{m}{2} \rfloor)} = c_2^{(\lfloor \frac{m}{2} \rfloor)}$, which is a stable coloring for the whole graph, as the color partition is not refined anymore.

The asymmetric case Without loss of generality, we can assume $m = \text{length}(c_1^{(t)}) \neq \text{length}(c_2^{(t)}) = m+h$ for some $h > 0$. We also assume for now that m is odd (the case m even will be briefly discussed later). We extend the HASH function from Lemma 3.4 to colors $j,k > \lfloor \frac{m}{2} \rfloor$. For $j > \lfloor \frac{m}{2} \rfloor$ or $k > \lfloor \frac{m}{2} \rfloor$ we define

$$\begin{cases} \text{HASH}(0, \{\!\{j,k\}\!\}) = \infty & \\ \text{HASH}(i, \{\!\{j,k\}\!\}) = \lfloor \frac{m}{2} \rfloor + i & \text{if } j \neq k,\ i \le \lfloor \frac{m}{2} \rfloor \\ \text{HASH}(i, \{\!\{j,k\}\!\}) = \lfloor \frac{m}{2} \rfloor + i + 1 & \text{if } j = k,\ i \le \lfloor \frac{m}{2} \rfloor \end{cases}.$$

Running in parallel the 1–WL test on the two cycles, computing the coloring vectors $c_1^{(\lfloor \frac{m}{2} \rfloor+1)}$ and $c_2^{(\lfloor \frac{m}{2} \rfloor+1)}$ up to iteration $\lfloor \frac{m}{2} \rfloor + 1$, for $i = \lfloor \frac{m}{2} \rfloor + 1$ we have $c_2(i) = \lfloor \frac{m}{2} \rfloor + 1$. Therefore, given the extension of the HASH function just provided, this new color starts to backpropagate on the indices $i < \lfloor \frac{m}{2} \rfloor + 1$, $i > m - h - \lfloor \frac{m}{2} \rfloor - 1$ until it reaches the index 0. As a consequence, it exists an iteration index T such that $c_2^{(T)}(0) = \text{HASH}(0, \{\!\{j,k^*\}\!\})$ with $k^* > \lfloor \frac{m}{2} \rfloor$ and, finally, $c_2^{(T)}(0) = \infty$, giving $c_1^{(T)}(0) \neq c_2^{(T)}(0)$, as claimed.

The case in which m is even works analogously, but we have to modify the HASH function in a different way to preserve injectivity. In particular, for $j,k \le m/2$, we define

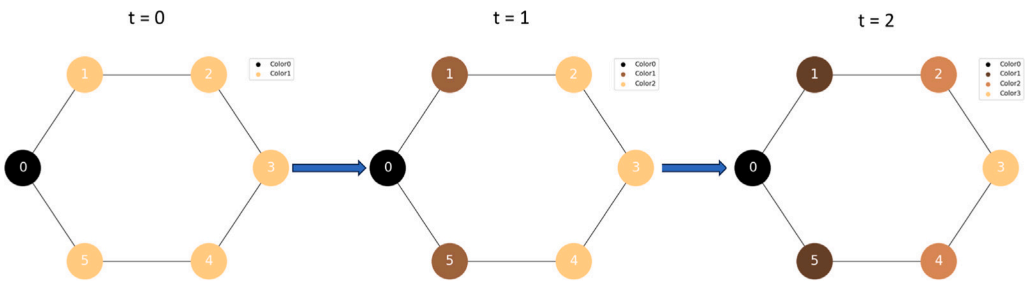
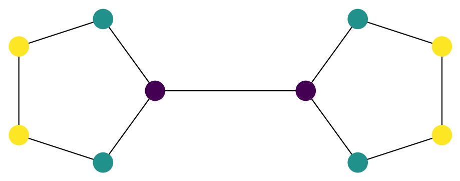
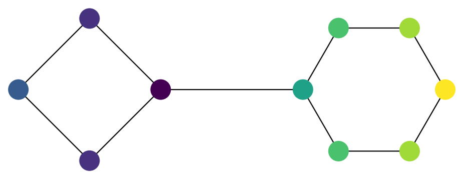


Fig. 2. Graphical illustration of Lemma 3.4: a 6-cycle reaches a stable coloring in $\lfloor \frac{6}{2} \rfloor = 3$ steps with $\lfloor \frac{6}{2} \rfloor + 1 = 4$ colors. Numbers are used to identify nodes.



(a) Stable 1-WL coloring for $[5, 5]$.



(b) Stable 1-WL coloring for $[4, 6]$.

Fig. 3. Stable 1-WL coloring for different types of dicyclic graphs: as stated in Theorem 3.5, 3-degree nodes have the same color in symmetric dicyclic graphs, and different color in the asymmetric ones.

$$\begin{cases} \text{HASH}(i, \{\!\{j, k\}\!\}) = \frac{m}{2} & \text{if } j = k, i = \frac{m}{2} \\ \text{HASH}(i, \{\!\{j, k\}\!\}) = \frac{m}{2} + 1 & \text{if } j \neq k, i = \frac{m}{2} \end{cases}.$$

This concludes the proof. □

Theorem 3.5 establishes in a deterministic way the power of the 1–WL test in terms of distinguishing between symmetric and asymmetric dicyclic graphs, given a sufficient number of iterations directly linked with the maximum cycle length in the considered domain. Examples of 1-WL stable colorings on dicyclic graphs are presented in Fig. 3.

Employing well-known results in the literature concerning the expressive power of GNNs (see Morris et al., 2019; Xu et al., 2018 and in particular Theorem 2.1), we can prove the main result of this subsection on the classification power of GNNs on the domain of dicyclic graphs.

Corollary 3.6 (*GNNs Can Classify Symmetric Dicyclic Graphs*)**.** *There exist a GNN of the form* (2) *and a* READOUT *function able to classify symmetric dicyclic graphs.*

Proof. Let $[m, n]$ be a dicyclic graph and $c^{(T)}$ be the stable coloring of $[m, n]$ produced by the 1-WL test with initial uniform coloring. By Theorem 3.5 the graph can be correctly classified by the 1–WL test, i.e., by its stable coloring. Using Theorem 2.1, a GNN f_Θ exists such that f_Θ can learn the stable coloring for each input graph for each iteration step t. Let $c^{(T)}$ be the stable coloring computed by a GNN for a dicyclic graph $[m, n]$. Let (u, v) be the 3-degree nodes of the dicyclic graph. Then, the READOUT can be modeled as

$$\text{READOUT}(c^{(T)}) = \begin{cases} 1 & \text{if } c^{(T)}(u) = c^{(T)}(v) \\ 0 & \text{otherwise} \end{cases}.$$

With such a READOUT, the GNN assigns the correct rating to the dicyclic graph (i.e., 1 if the graph is symmetric, 0 otherwise). □

Remark 3.7 (*The Gap Between Theory and Practice In* Corollary 3.6)**.** Corollary 3.6 shows that GNNs are powerful enough to match the 1–WL test's expressive power for the classification of symmetric dicyclic graphs (as established by Theorem 3.5). However, it is worth underlining that this result only proves the *existence* of a GNN model able to perform this task. In contrast to the results presented in Section 3.1, this corollary does not mention any training procedure. Nevertheless, the numerical experiments in Section 4.3 show that GNNs able to classify symmetric dicyclic graphs *can* be trained in practice, albeit achieving generalization outside the training set is not straightforward and depends on the GNN architecture.

4. Numerical results

This section presents the results of experimental tasks designed to validate our theorems. We analyze the consistency between theoretical and numerical findings, highlighting the significance of specific hypotheses, and addressing potential limitations of the theoretical results.

4.1. Experimental setup

We take in account two different models for our analysis:

- The Global Additive Pooling GNN (*Gconv-glob*) applies a sum pooling at the end of the message-passing convolutional layers (Hamilton et al., 2017). In the case of the 2-letter words setting, the resulting vector $h_{\text{glob}} \in \mathbb{R}^h$ undergoes processing by a linear layer, while in the dicyclic graphs setting, an MLP is employed. A sigmoid activation function is applied at the end.
- The Difference GNN (*Gconv-diff*), takes the difference between the hidden states of the two nodes in the graph (in the 2-letter words setting) or the difference between the hidden states of the 3-degree nodes (in the dicyclic graphs setting) after the message-passing convolutional layers. The resulting vector $h_{\text{diff}} \in \mathbb{R}^h$ is then fed into a final linear layer, followed by the application of a sigmoid activation function.

The choice of the last READOUT part is driven by empirical observation on their effectiveness on the two different tasks.

Training is performed on an Intel(R) Core(TM) i7-9800X processor running at 3.80 GHz using 31 GB of RAM along with a GeForce GTX 1080 Ti GPU unit. The Python code is available at https://github.com/AleDinve/gnn_identity_effects.git.

4.2. Case study #1: two-letter words

To validate Theorem 3.1, we consider a classification task using the two-letter word identity effect problem described in Section 3.1.1, following the experimental setup presented in Brugiapaglia et al. (2022).

4.2.1. Task and datasets

In accordance with the setting of Section 3.1.1, each word is represented as a graph consisting of two nodes connected by a single unweighted and undirected edge (see Fig. 1). Each node is assigned a node feature $x \in \mathbb{R}^{26}$, corresponding to a letter's encoding.

The training set D_{train} includes all two-letter words composed of any English alphabet letters except Y and Z. The test set D_{test} is a set of two-letter words where at least one of the letters is chosen from Y, Z. Specifically, we consider $D_{\text{test}} = \{\text{YY}, \text{ZZ}, \text{YZ}, \text{ZT}, \text{EY}, \text{SZ}\}$.

4.2.2. Vertex feature encodings

In our experiments, we consider four different encodings of the English alphabet, following the framework outlined in Section 3.1. Each encoding consists of a set of vectors drawn from $\mathbb{R}^{26}$.

- *One-hot encoding*: This encoding assigns a vector from the canonical basis to each letter: A is encoded as e_1, B as e_2, …, and Z as e_{26}.
- *Haar encoding*: This encoding assigns to each letter the columns of a 26×26 orthogonal matrix drawn from the orthogonal group O(26) using the Haar distribution (Mezzadri, 2007).
- *Distributed encoding*: This encoding assigns a random combination of 26 bits to each letter. In this binary encoding, only j bits are set to 1, while the remaining $26 - j$ bits are set to 0. In our experiments, we set $j = 6$.
- *Gaussian encoding*: This encoding assigns samples from the multivariate normal distribution $\mathcal{N}(0, I)$, where $0 \in \mathbb{R}^n$ and $I \in \mathbb{R}^{n\times n}$. In our experiments, we set $n = 16$.

Observe that only the one-hot and the Haar encodings are orthogonal (see Section 3.1.1) and hence satisfy the assumption of Theorem 3.3. On the other hand, the distributed and the Gaussian encodings do not fall within the setting of Theorem 3.3.

We run 40 trials for each model (i.e., Gconv-glob or Gconv-diff, defined in Section 4.1) with l layers (ranging from 1 to 3). In each trial, a different training set is randomly generated. The models are trained for 5000 epochs using the Adam optimizer with a learning rate of $\lambda = 0.0025$, while minimizing the binary cross-entropy loss. The hidden state dimension is set to $d = 64$, and Rectified Linear Units (ReLUs) are used as activation functions.

The numerical results are shown in Figs. 4–5, where we propose two different types of plots:

- On the top row, we compare the ratings obtained using the four adopted encodings. The first two words, AA and a randomly generated word with nonidentical letters, denoted xy, are selected from the training set to showcase the training accuracy. The remaining words are taken from D_{test}, allowing assessment of the generalization capabilities of the encoding scheme outside the training test. The bars represent the mean across trials, while the segments at the center of each bar represent the standard deviation.
- On the bottom row, we show loss functions with respect to the test set over the training epochs for each encoding. The lines represent the average, while the shaded areas represents the standard deviation.

Our numerical findings indicate that the rating impossibility theorem holds true for the one-hot encoding and the Haar encoding. However, notable differences in behavior emerge for the other two encodings. The 6-bit distributed encoding exhibits superior performance across all experiments, demonstrating higher rating accuracy and better loss convergence. The Gaussian encoding yields slightly inferior results, yet still showcases some generalization capability. It is important to note that despite variations in experimental settings such as architecture and optimizer (specifically, the use of ReLU activations and the Adam optimizer), the divergent behavior among the considered encodings remains consistent. This highlights the critical role of the transformation matrix T_2 within the hypothesis outlined in Theorem 3.3. It is interesting to notice that increasing the number of layers contributes to the so-called *oversmoothing effect* (Cai & Wang, 2020; Oono & Suzuki, 2019): many message passing iterations tend to homogenize information across the nodes, generating highly similar features.

4.3. Case study #2: dicyclic graphs

We now consider the problem of classifying unlabeled symmetric dicyclic graphs, introduced in Section 3.2. In Corollary 3.6 we proved the existence of GNNs able to classify symmetric dicyclic graphs. In this section, we assess whether such GNNs can be computed via training (see also Remark 3.7). With this aim, we consider two experimental settings based on different choices of training and test set: an *extraction task* and an *extrapolation task*, summarized in Figs. 8 and 10, respectively, and described in detail below. Each task involves running 25 trials for the Gconv-glob and Gconv-diff models defined in Section 4.1. The number of layers in each model is determined based on the specific task.

The models are trained over 5000 epochs using a learning rate of $\lambda = 0.001$. We employ the Adam optimizer, minimizing the binary crossentropy, and incorporate the AMSGrad fixer (Reddi, Kale, & Kumar, 2019) to enhance training stability due to the large number of layers. Labels are all initialized uniformly as $h_v^{(0)} = 1$ for each node in each graph. The hidden state dimension is set to $d = 100$, and ReLU activation functions are utilized.

The results presented in Figs. 6, 8, and 10 should be interpreted as follows: each circle represents a dicyclic graph $[m, n]$; the color of the circle corresponds to the rating, while the circle's radius represents the standard deviation.

4.3.1. 1–WL test performance

In Theorem 3.5 we showed that the 1–WL test can classify symmetric dicyclic graphs. This holds true regardless of the length of the longer cycle, provided that a sufficient number of iterations is performed. The results in Fig. 6 show that the 1–WL test achieves indeed perfect classification accuracy in $n_{\max}$ iterations, where $n_{\max}$ is the maximum length of a cycle in the dataset, in accordance with Theorem 3.5.

4.3.2. Extraction task

In this task, we evaluate the capability of GNNs to generalize to unseen data, specifically when the minimum length of cycles in the test dataset is smaller than the maximum length of those in the training dataset. More specifically, the training set D_{train} consists of pairs $[m, n]$ where $3 \le m, n \le n_{\max}$ and $m, n \ne k$ with $3 \le k \le n_{\max}$, while the test set D_{test} comprises pairs $[k, a]$ with $3 \le a \le n_{\max}$. Fig. 7 illustrates this setting.

In our experiments, we set $n_{\max} = 8$ and consider k values of 7, 6, and 5. In this setting, $|D_{\text{test}}| = (8-2)\cdot 2 - 1 = 11$ and $|D_{\text{train}}| = (8-2)^2 - |D_{\text{test}}| = 25$. The number of GNN layers is $l = n_{\max}$. The numerical results are presented in Fig. 8. We observe that the Gconv-diff model achieves perfect performance in our experiments (standard deviation values are not reported because they are too low), showing consistence with the theoretical setting. On the other hand, the Gconv-glob model demonstrates good, but not perfect, performance on the test set. A critical point in our numerical examples seems to be $k = 5$, which falls in the middle range between the minimum and maximum cycle lengths in the training set (3 and 8, respectively). This particular value is closer to the minimum length, indicating a relatively unbalanced scenario.

Overall, the different performance of Gconv-diff and Gconv-glob on the extraction task shows that, despite the theoretical existence result proved in Corollary 3.6, the choice of architecture is crucial for achieving successful generalization.

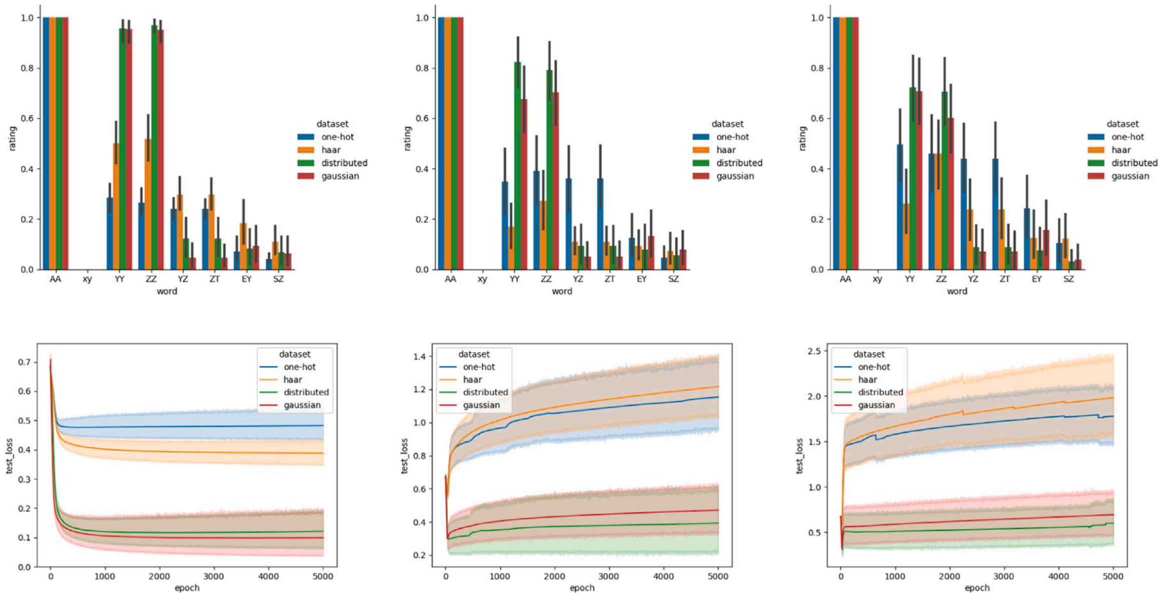


Fig. 4. Numerical results for the rating task on the two-letter words dataset using Gconv-glob with $l = 1, 2, 3$ layers. Rating should be equal to 1 if words are composed by identical letters, 0 otherwise. The distributed and Gaussian encodings, which deviate from the framework outlined in Theorem 3.1, exhibit superior performance compared to the other encodings. The other encodings makes the transformation matrix orthogonal and symmetric, being themselves orthogonal encodings.

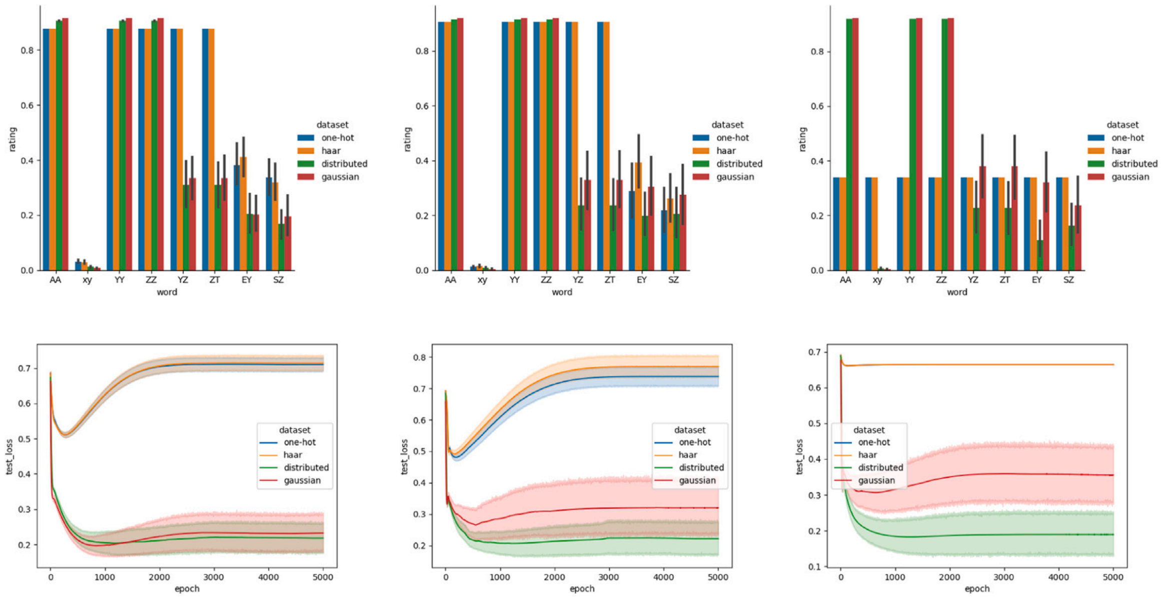


Fig. 5. Numerical results for the rating task on the two-letter words dataset using Gconv-diff with $l = 1, 2, 3$ layers. The same observations to those in Fig. 4 can be made here as well.

4.3.3. *Extrapolation task*

In this task, we assess GNNs' ability to generalize to unseen data with cycle lengths exceeding the maximum length in the training dataset. Specifically, the training set D_{train} comprises pairs $[m, n]$ where $3 \leq m, n \leq n_{\max}$, while the test set D_{test} consists of pairs $[n_{\max} + k, n']$ with $0 < k \leq g$ and $3 \leq n' \leq n_{\max} + g$. Fig. 9 illustrates the extrapolation task.

In our experiments, we set $n_{\max} = 8$ and consider values of g as 1, 2, and 3. The number of GNN layers is $l = n_{\max} + g$. Therefore, $|D_{\text{train}}| = (8-2)^2 = 36$, $|D_{\text{test},g=1}| = (9-2)\cdot 2-1 = 13$, $|D_{\text{test},g=2}| = (10-2)\cdot 4-4 = 28$ and $|D_{\text{test},g=3}| = (11-2)\cdot 6-9 = 45$. Numerical results are presented in Fig. 10. In the extraction task, both models achieved perfect training accuracy. Conversely, in the extrapolation task, the Gconv-glob model struggles to classify the training set accurately, especially when the number of layers is equal to 9. This behavior may be attributed to the homogeneous nature of sum pooling at the end of the message passing, as it does not take into account the role of 3-degree nodes (which play a key role in our theory, as illustrated by Theorem 3.5 and Corollary 3.6).

On the other hand, the Gconv-diff model consistently achieves perfect training accuracy over the training set and achieves perfect generalization for $g = 1$, showing once again the importance of architecture choice in practice. However, when $g \geq 2$ there is a noticeable region of misclassification for pairs $[m, n]$ where $m, n \geq n_{\max}$. This behavior could be explained by the limited capacity of the hidden states, but the optimization process might also play a significant role. Moreover, for $g \geq 2$ the numerical results of the extrapolation task resemble the rating impossibility phenomenon observed in the two-letter words framework. However, it is important to note that, at least for the Gconv-diff model, we observe significantly different ratings between graphs $[m, n_{\max} + g]$ where $m < n_{\max}$ and graphs $[n_{\max} + i, n_{\max} + j]$ with $i, j > 0$. In contrast,

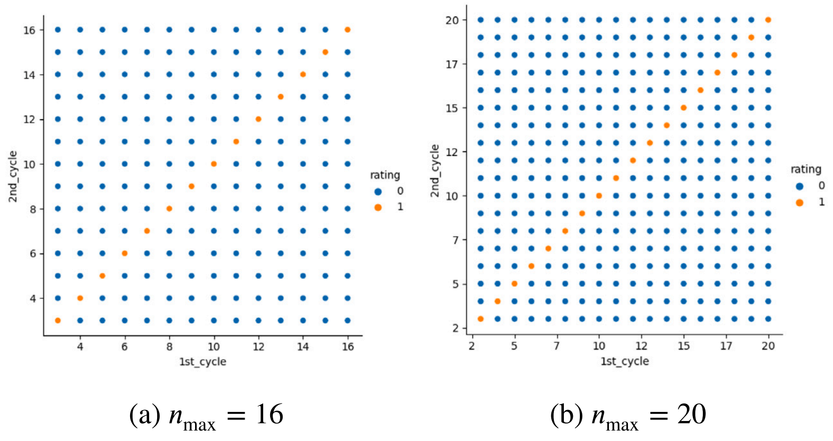


Fig. 6. Perfect classification of symmetric dicyclic graphs by $n_{\max}$ iterations of the 1-WL test.

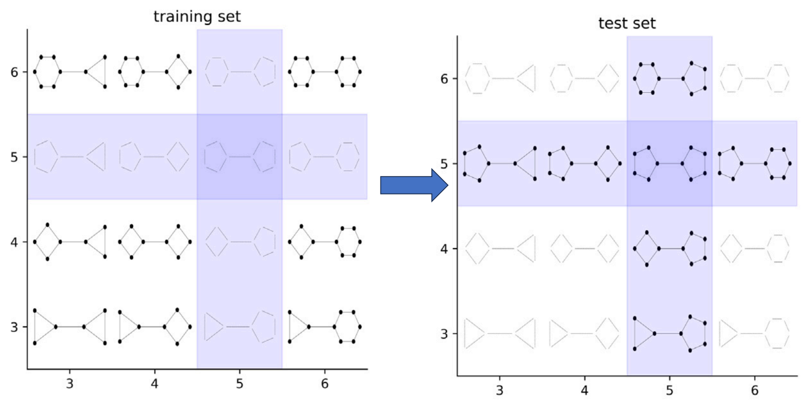


Fig. 7. Graphical illustration of the extraction task. In this example, $n_{\max} = 6$ and $k = 5$.

in the two-letter words framework ratings typically do not exhibit such a consistent and distinguishable pattern.

5. Conclusions

This work extensively investigates the generalization capabilities of GNNs when learning identity effects through a combination of theoretical and experimental analysis. From the theoretical perspective, in Theorem 3.3 we established that GNNs, under mild assumptions, cannot learn identity effects when orthogonal encodings are used in a specific two-letter word classification task. On the positive side, in Corollary 3.6 we showed the existence of GNNs able to successfully learn identity effects on dicyclic graphs, thanks to the expressive power of the Weisfeiler–Lehman test (see Theorem 3.5). The experimental results strongly support these theoretical findings and provide valuable insights into the problem. In the case of two-letter words, our experiments highlight the key influence of encoding orthogonality on misclassification behavior. Our experiments on dicyclic graphs demonstrate the importance of architecture choice in order to achieve generalization.

Several directions of future research naturally stem from our work. First, while Theorem 3.3 identifies sufficient conditions for rating impossibility, it is not known whether (any of) these conditions are also necessary. Moreover, numerical experiments on two-letter words show that generalization outside the training set is possible when using nonorthogonal encodings; justifying this phenomenon from a theoretical perspective is an open problem. On the other hand, our numerical experiments on dicyclic graphs show that achieving generalization depends on choice of the architecture; this suggests that rating impossibility theorems might hold under suitable conditions on the GNN architecture in that setting. Another interesting open problem is the evaluation of GNNs' expressive power on more complex graph domains. In particular, conducting extensive experiments on molecule analyses mentioned in Section 1, which naturally exhibit intricate structures, could provide valuable insights into modern chemistry and drug discovery applications.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

GAD is partially supported by the Gruppo Nazionale per il Calcolo Scientifico (GNCS) of the Istituto Nazionale di Alta Matematica (INdAM). SB acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) through grant RGPIN-2020-06766 and the Fonds de Recherche du Québec Nature et Technologies (FRQNT), Canada through grant 313276. MR acknowledges the support of NSERC, Canada through grant 263344. The authors are thankful to Aaron Berk for providing feedback on a preliminary version of this manuscript and to Kara Hughes for fruitful discussions on applications to the molecule domain.

Data availability

Code is available on GitHub repo, mentioned in the paper.

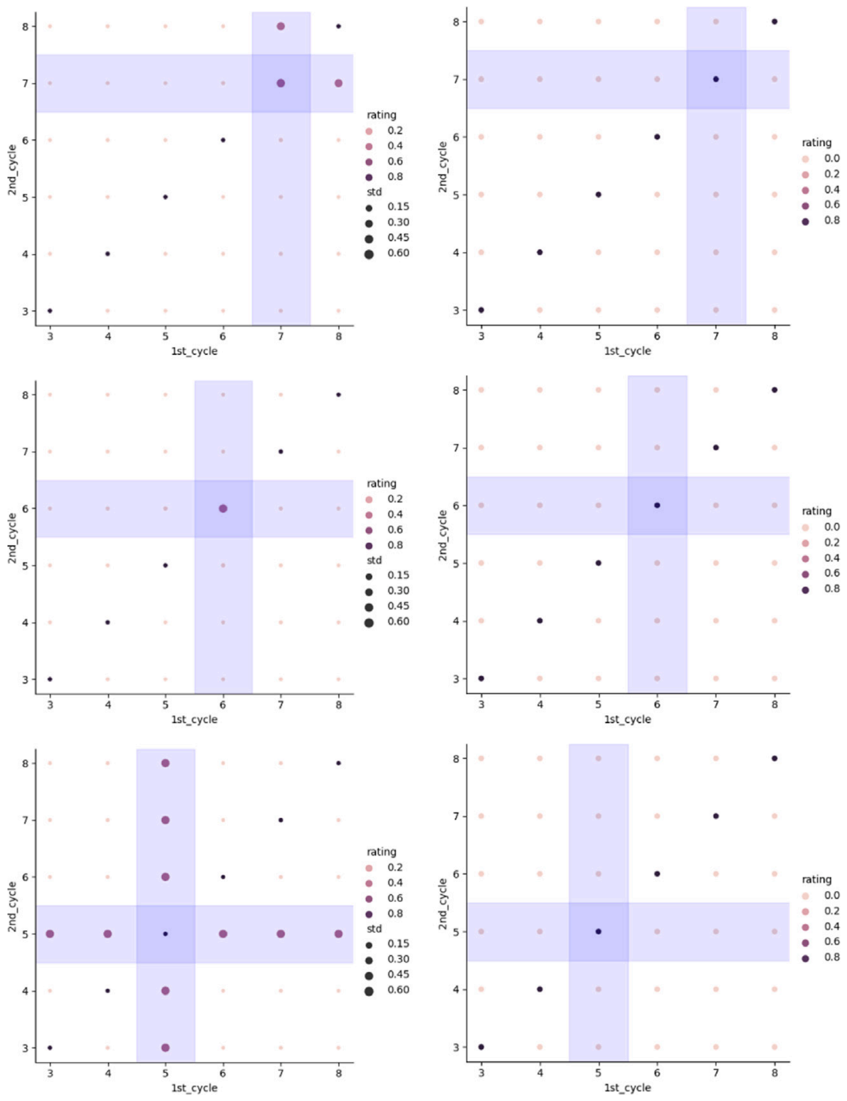


Fig. 8. Extraction task performed by different GNN models, namely Gconv-glob (left) and Gconv-diff (right). We set $n_{\max} = 8$, $l = 8$ and, from top to bottom, $k = 7, 6, 5$.

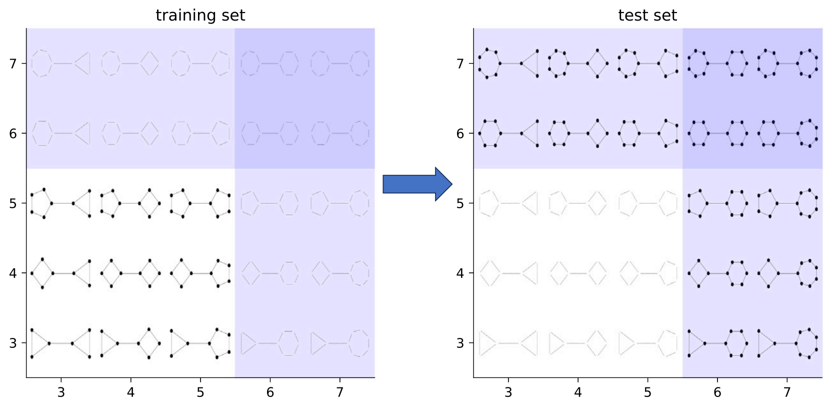


Fig. 9. Graphical illustration of the extrapolation task. In this example, $n_{\max} = 5$ and $g = 2$.

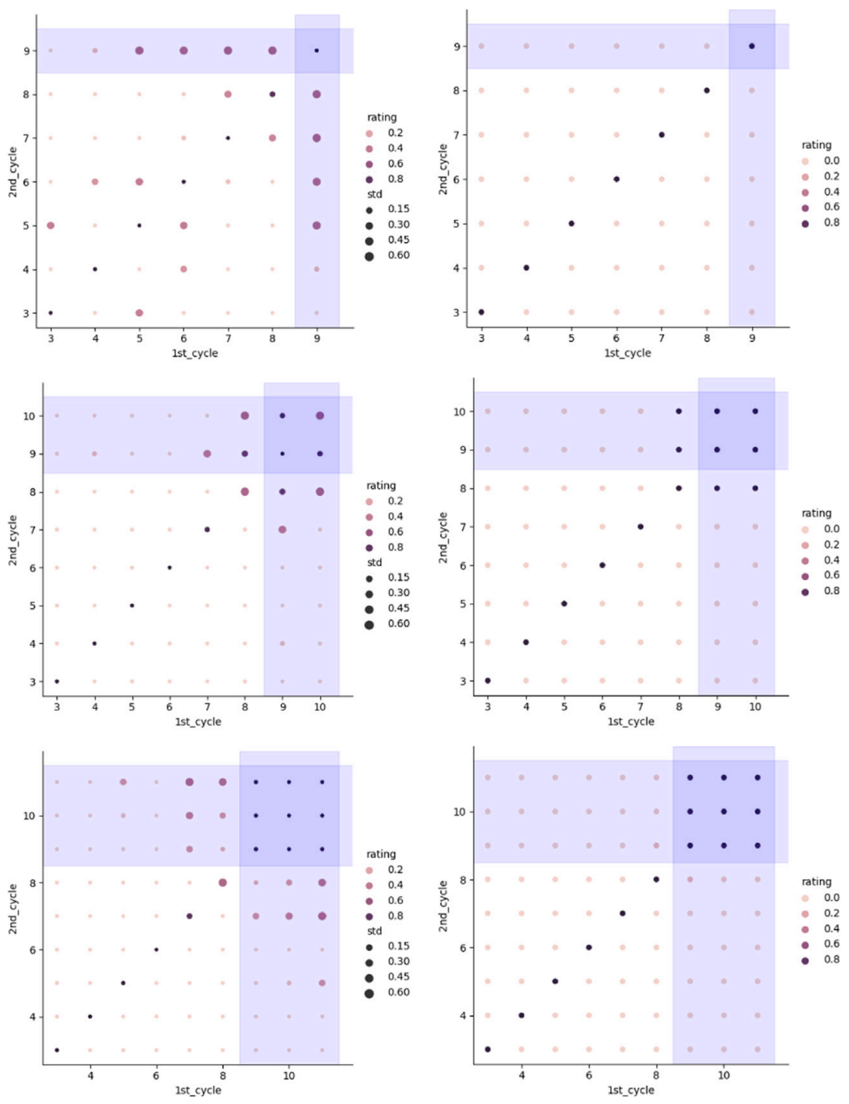


Fig. 10. Extrapolation task performed by different GNN models, namely Gconv-glob (left) and Gconv-diff (right). We set $n_{\max} = 8$, $l = 8$ and, from top to bottom, $(l, g) = (9, 1), (10, 2), (11, 3)$.

References

Azizian, W., & Lelarge, M. (2020). Expressive power of invariant and equivariant Graph Neural Networks. arXiv preprint arXiv:2006.15646.

Babai, L., & Kucera, L. (1979). Canonical labelling of graphs in linear average time. In *20th annual symposium on foundations of computer science* (pp. 39–46). IEEE.

Bartlett, P. L., & Mendelson, S. (2002). Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, *3*(Nov), 463–482.

Benua, L. (1995). Identity effects in morphological truncation. In J. Beckman, S. Urbanczyk, & L. Walsh (Eds.), *Papers in optimality theory* (pp. 77–136). Amherst, MA: GLSA (Graduate Linguistic Student Association), Dept. of Linguistics, University of Massachusetts.

Bianchini, M., & Scarselli, F. (2014). On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(8), 1553–1565.

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., et al. (2021). Weisfeiler and Lehman go cellular: CW networks. *Advances in Neural Information Processing Systems*, *34*, 2625–2640.

Bodnar, C., Frasca, F., Wang, Y., Otter, N., Montufar, G. F., Lio, P., et al. (2021). Weisfeiler and Lehman go topological: Message passing simplicial networks. In *International conference on machine learning* (pp. 1026–1037). Proceedings of Machine Learning Research.

Bongini, P., Bianchini, M., & Scarselli, F. (2021). Molecular generative graph neural networks for drug discovery. *Neurocomputing*, *450*, 242–252.

Brugiapaglia, S., Liu, M., & Tupper, P. (2022). Invariance, encodings, and generalization: learning identity effects with neural networks. *Neural Computation*, *34*(8), 1756–1789.

Bunker, P. R., & Jensen, P. (2006). *Molecular symmetry and spectroscopy, vol. 46853*. NRC research Press.

Cai, C., & Wang, Y. (2020). A note on over-smoothing for graph neural networks. arXiv preprint arXiv:2006.13318.

Chamberlain, B., Rowbottom, J., Gorinova, M. I., Bronstein, M., Webb, S., & Rossi, E. (2021). GRAND: Graph neural diffusion. In *International conference on machine learning* (pp. 1407–1418). Proceedings of Machine Learning Research.

D'Inverno, G. A., Bianchini, M., Sampoli, M. L., & Scarselli, F. (2024). On the approximation capability of GNNs in node classification/regression tasks. *Soft Computing*, *28*(13), 8527–8547.

Fan, W., Ma, Y., Li, Q., Wang, J., Cai, G., Tang, J., et al. (2020). A graph neural network framework for social recommendations. *IEEE Transactions on Knowledge and Data Engineering*, *34*(5), 2033–2047.

Gallagher, G. (2013). Learning the identity effect as an artificial language: bias and generalisation. *Phonology*, *30*(2), 253–295.

Garg, V., Jegelka, S., & Jaakkola, T. (2020). Generalization and representational limits of Graph Neural Networks. In *International conference on machine learning* (pp. 3419–3430). Proceedings of Machine Learning Research.

Ghomeshi, J., Jackendoff, R., Rosen, N., & Russell, K. (2004). Contrastive focus reduplication in English (the salad-salad paper). *Natural Language & Linguistic Theory*, *22*, 307–357.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th international conference on machine learning, vol. 70* (pp. 1263–1272).

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th international conference on artificial intelligence and statistics* (pp. 249–256). JMLR Workshop and Conference Proceedings.

Golowich, N., Rakhlin, A., & Shamir, O. (2018). Size-independent sample complexity of neural networks. In *Conference on learning theory* (pp. 297–299). Proceedings of Machine Learning Research.

Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, *14*(3), 1–159.

Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, *30*.

Jiang, W., & Luo, J. (2022). Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, Article 117921.

Keriven, N., & Peyré, G. (2019). Universal invariant and equivariant Graph Neural Networks. *Advances in Neural Information Processing Systems*, *32*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

Leman, A., & Weisfeiler, B. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, *2*(9), 12–16.

Liao, R., Urtasun, R., & Zemel, R. (2020). A pac-bayesian approach to generalization bounds for Graph Neural Networks. arXiv preprint arXiv:2012.07690.

Liebman, J. F., & Greenberg, A. (1976). A survey of strained organic molecules. *Chemical Reviews*, *76*(3), 311–365.

Longa, A., Lachi, V., Santin, G., Bianchini, M., Lepri, B., Lio, P., et al. (2023). Graph neural networks for temporal graphs: state of the art, open challenges, and opportunities. *Transactions on Machine Learning Research*, [ISSN: 2835-8856] https://openreview.net/forum?id=pHCdMat0gI.

Malekzadeh, M., Hajibabaee, P., Heidari, M., Zad, S., Uzuner, O., & Jones, J. H. (2021). Review of graph neural network in text classification. In *2021 IEEE 12th annual ubiquitous computing, electronics & mobile communication conference* (pp. 0084–0091). IEEE.

Marcus, G. F. (2003). *The algebraic mind: Integrating connectionism and cognitive science*. MIT Press.

Marcus, G. F., Vijayan, S., Bandi Rao, S., & Vishton, P. M. (1999). Rule learning by seven-month-old infants. *Science*, *283*(5398), 77–80.

Maskey, S., Levie, R., Lee, Y., & Kutyniok, G. (2022). Generalization analysis of message passing neural networks on large random graphs. In *Advances in neural information processing systems*.

Mezzadri, F. (2007). How to generate random matrices from the classical compact groups. *Notices of the American Mathematical Society*, *54*(5), 592–604.

Morris, C., Geerts, F., Tönshoff, J., & Grohe, M. (2023). WL meet VC. In *International conference on machine learning* (pp. 25275–25302). Proceedings of Machine Learning Research.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., et al. (2019). Weisfeiler and lehman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence, vol. 33, no. 01* (pp. 4602–4609).

Oono, K., & Suzuki, T. (2019). Graph neural networks exponentially lose expressive power for node classification. arXiv preprint arXiv:1905.10947.

Paschen, L. (2021). Trigger poverty and reduplicative identity in Lakota. *Natural Language & Linguistic Theory*, 1–37.

Pettinari, C., & Santini, C. (2017). IR and Raman spectroscopies of inorganic, coordination and organometallic compounds. In J. Lindon, G. Tranter, & D. W. Koppenaal (Eds.), *Encyclopedia of spectroscopy and spectrometry* ((3rd ed.)). (pp. 347–358). Oxford: Academic Press, http://dx.doi.org/10.1016/B978-0-12-803224-4.00338-1.

Reddi, S. J., Kale, S., & Kumar, S. (2019). On the convergence of Adam and beyond. arXiv preprint arXiv:1904.09237.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, (1), 61–80.

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

Shi, L., Zhang, Y., Cheng, J., & Lu, H. (2019). Skeleton-based action recognition with directed graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 7912–7921).

Suárez, L. E., Richards, B. A., Lajoie, G., & Misic, B. (2021). Learning function from structure in neuromorphic networks. *Nature Machine Intelligence*, *3*(9), 771–786.

Tupper, P., & Shahriari, B. (2016). Which learning algorithms can generalize identity-based rules to novel inputs? In *Proceedings of the 28th annual meeting of the cognitive science society*.

Vapnik, V., Levin, E., & Le Cun, Y. (1994). Measuring the VC-dimension of a learning machine. *Neural Computation*, *6*(5), 851–876.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention etworks. arXiv preprint arXiv:1710.10903.

Vershynin, R. (2018). *High-dimensional probability: An introduction with applications in data science, vol. 47*. Cambridge University Press.

Wieder, O., Kohlbacher, S., Kuenemann, M., Garon, A., Ducrot, P., Seidel, T., et al. (2020). A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, *37*, 1–12.

Wu, Q., & Wang, Q. (2010). Natural language processing based detection of duplicate defect patterns. In *2010 IEEE 34th annual computer software and applications conference workshops* (pp. 220–225). IEEE.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks? arXiv preprint arXiv:1810.00826.

Zhang, R., Zou, Y., & Ma, J. (2019). Hyper-SAGNN: a self-attention based Graph Neural Network for hypergraphs. arXiv preprint arXiv:1911.02613.