A Principled Approach to Analyze Expressiveness and Accuracy of Graph Neural Networks

Asma Atamna, Nataliya Sokolovska, Jean-Claude Crivello

A Principled Approach to Analyze Expressiveness and Accuracy of Graph Neural Networks

Asma Atamna,[1] **Nataliya Sokolovska,**[2] **Jean-Claude Crivello**[1]
[1]ICMPE (UMR 7182), CNRS, University of Paris-Est, Thiais, France
name@icmpe.cnrs.fr
[2]NutriOmics, INSERM, Sorbonne University, Paris, France
nataliya.sokolovska@sorbonne-universite.fr

Abstract

Graph neural networks (GNNs) have known an increasing success recently, with many GNN variants achieving state-of-the-art results on node and graph classification tasks. The proposed GNNs, however, often implement complex node and graph embedding schemes, which makes it challenging to explain their performance. In this paper, we investigate the link between a GNN's *expressiveness*, that is, its ability to map different graphs to different representations, and its generalization performance in a graph classification setting. In particular, we propose a principled experimental procedure where we (i) define a practical measure for expressiveness, (ii) introduce an expressiveness-based loss function that we use to train a simple yet practical GNN that is permutation-invariant, (iii) illustrate our procedure on benchmark graph classification problems and on an original real-world application. Our results reveal that expressiveness alone does not guarantee a better performance, and that a powerful GNN should be able to produce graph representations that are *well separated* with respect to the class of the corresponding graphs.

1 Introduction

Many real-world data present an inherent structure and can be modelled as sequences, graphs, or hypergraphs (Collobert et al. 2011; Min et al. 2017; Duvenaud et al. 2015; Bojchevski et al. 2018). Graph-structured data in particular are very common in practice and are at the heart of this work.

We consider the problem of graph classification. That is, given a set $\mathcal{G} = \{G_i\}_{i=1}^m$ of arbitrary graphs G_i and their respective labels $\{y_i\}_{i=1}^m$, where $y_i \in \{1, \dots, C\}$ and $C \geq 2$ is the number of classes, we aim at finding a mapping $f_\theta(G) : \mathcal{G} \to \{1, \dots, C\}$, that minimizes the classification error, where θ denotes the parameters to optimize.

Recent years have known a surge of interest in graph neural networks (GNNs) and their deep learning variants, the graph convolutional networks (GCNs) (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2017; Atwood and Towsley 2016; Duvenaud et al. 2015; Zhang et al. 2018; Niepert, Ahmed, and Kutzkov 2016; Gilmer et al. 2017; Schlichtkrull et al. 2018). GNNs learn latent node representations by recursively aggregating the neighboring node features for each node, thereby capturing the structural information of a node's neighborhood.

Despite the profusion of GNN variants, some of which achieve state-of-the-art results on tasks like node classification, graph classification, and link prediction, GNNs remain very little studied. In particular, it is often unclear what a GNN has learned and how the learned graph (or node) mapping influences its generalization performance. In a recent work, (Xu et al. 2019) present a theoretical framework to analyze the expressive power of GNNs, where a GNN's *expressiveness* is defined as its ability to compute different graph representations for different graphs. Theoretical conditions under which a GNN is maximally expressive are derived. Although it is reasonable to assume that a higher expressivess would result in a higher test accuracy on classification tasks, this link has not been explicitly studied so far.

In this paper, we design a principled experimental procedure to analyze the link between expressiveness and the test accuracy of GNNs. In particular,

- We define a practical measure to estimate the expressiveness of GNNs;
- We use this measure to define a new penalized loss function that allows training GNNs with varying expressive power;
- We present a simple yet practical architecture, the Simple Permutation-Invariant Graph Convolutional Network (SPI-GCN), on which we illustrate our experimental framework.

We also present an original graph data set of metal hydrides that we use along with benchmark graph data sets to evaluate SPI-GCN.

This paper is organized as follows. Section 2 discusses the related work. Section 3 introduces preliminary notations and concepts related to graphs and GNNs. In Section 4, we introduce our graph neural network, SPI-GCN. In Section 5, we present a practical expressiveness estimator and a new expressiveness-based loss function as part of our experimental framework. Section 6 presents our results and Section 7 concludes the paper.

2 Related Work

Two main supervised learning approaches for graph-structured data include graph kernels and graph neural networks.

Graph kernels (Haussler 1999; Kriege, Johansson, and Morris 2019) define a similarity measure on graphs that allows the application of kernel methods to graph classification. The similarity between two graphs is usually computed by decomposing the graphs into substructures, then comparing the latter pairwise. This procedure can be expensive, especially on large graphs. An effective class of graph kernels are the Weisfeiler-Lehman (WL) kernels (Shervashidze et al. 2011) that implement a feature extraction mechanism based on the WL algorithm (Weisfeiler and Lehman 1968) for graph isomorphism test. Graph kernels have been the state-of-the-art in graph classification; their main drawback, however, is their computational inefficiency. In particular, the training complexity of graph kernels is at least quadratic in the number of graphs (Shervashidze et al. 2011).

Graph neural networks (GNNs) were first introduced in (Gori, Monfardini, and Scarselli 2005; Scarselli et al. 2009). They learn latent node representations by iteratively aggregating neighborhood information for each node. More recent graph convolutional networks (GCNs) generalize conventional convolutional neural networks to irregular graph domains. In (Kipf and Welling 2017), the authors present a GCN for node classification where the computed node representations can be interpreted as the graph coloring returned by the 1-dimensional WL algorithm. A related GCN that is invariant to node permutation is presented in (Zhang et al. 2018). The graph convolution operator is closely related to the one in (Kipf and Welling 2017), and the authors introduce a permutation-invariant pooling operator that sorts the convolved nodes before feeding them to a 1-dimensional classical convolution layer for graph-level classification. A popular graph deep learning approach is PATCHY-SAN (Niepert, Ahmed, and Kutzkov 2016). Its graph convolution operator extracts normalized local "patches" (neighborhood representations) of the graph which are then sorted and fed to a 1-dimensional traditional convolution layer for graph-level classification. The method, however, requires the definition of a node ordering and running the WL algorithm in a preprocessing step. On the other hand, the normalization of the extracted patches implies sorting the nodes again and using the external graph software NAUTY (Mckay and Piperno 2014).

Despite the success of GNNs, there are relatively few papers that analyze their properties, either mathematically or empirically. A notable exception is the recent work by (Xu et al. 2019) that studies the expressive power of GNNs. The authors prove that (i) GNNs are at most as powerful as the WL test in distinguishing graph structures and that (ii) if the graph function of a GNN—i.e. its graph embedding scheme—is injective, then the GNN is as powerful as the WL test. In another study (Chen, Bian, and Sun 2019), the authors present a simple neural network defined on a set of graph augmented features and show that their architecture can be obtained by linearizing graph convolutions in GNNs.

Our work is related to (Xu et al. 2019) in that we adopt the same definition of expressiveness, that is, the ability of a GNN to compute distinct graph representations for distinct input graphs. However, we go one step further and investigate how the graph function learned by GNNs affects their generalization performance. On the other hand, our SPI-GCN extends the GCN in (Kipf and Welling 2017) to graph-level classification. Our SPI-GCN is also related to (Zhang et al. 2018) in that we adopt the same graph convolution operator inspired by (Kipf and Welling 2017). Unlike (Zhang et al. 2018), however, our architecture does not require any node ordering, and we only use a simple multilayer perceptron (MLP) to perform classification.

3 Preliminaries

3.1 Matrix and Vector Notations

We denote matrices by capital letters and vectors by small letters. Scalars, on the other hand, are denoted by small italic letters. Consider a matrix M. m_i denotes its ith row and m_{ij} denotes the entry at its ith row and jth column. Its inverse matrix is denoted M^{-1}.

3.2 Some Graph Concepts

A graph G is a pair (V, E) of a set $V = \{v_1, \ldots, v_n\}$ of *vertices* (or *nodes*) v_i, and a set $E \subseteq V \times V$ of *edges* (v_i, v_j). In practice, a graph G is often represented by an *adjacency matrix* $\mathrm{A} \in \mathbb{R}^{n \times n}$ modelling the edges of the graph, where $n = |V|$ is the number of vertices, and such that $a_{ij} = 1$ if there is an edge between nodes v_i and v_j ($(v_i, v_j) \in E$) and $a_{ij} = 0$ otherwise. Edges can be either oriented, and we say that the graph is *directed*, or non-oriented, in which case we say that the graph is *undirected*. Note that the adjacency matrix of undirected graphs is symmetric. In an undirected graph, we say that two vertices v_i and v_j are *neighbors* if there exists an edge $(v_i, v_j) \in E$, and we denote $N(i)$ the *neighborhood* of v_i, i.e. the set of the indices of all neighbors of v_i. The number of neighbors, $|N(i)|$, of a node v_i is called the *degree* of v_i. In directed graphs, similar notions of *indegree* and *outdegree* exist. Finally, edges of the form (v_i, v_i), i.e. edges between a node and itself, are referred to as *self-loops*.

We assume that a graph G is characterized by a *node feature matrix* $\mathrm{X} \in \mathbb{R}^{n \times d}$, with d being the number of node features, in addition to its adjacency matrix A. Each row $\mathrm{x}_i \in \mathbb{R}^d$ of X contains the feature representation of a node v_i, where d is the dimension of the feature space. Since we only consider node features in this paper (as opposed to *edge features* for instance), we will refer to the node feature matrix X simply as the *feature matrix* in the rest of this paper.

We now introduce the notion of *graph isomorphism* in Definition 1.

Definition 1 (Graph Isomorphism)**.** Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a bijection $g : V_1 \to V_2$ such that every edge (u, v) is in E_1 if and only if the edge $(g(u), g(v))$ is in E_2.

Informally, Definition 1 states that two graphs are isomorphic if there exists a vertex permutation such that when applied to one graph, we recover the vertex and edge sets of the

other graph. That is, two graphs are isomorphic if they have the same structure independently of the vertex indexing.

The problem of determining whether two graphs are isomorphic is called the graph isomorphism (GI) problem and is an important one in graph and complexity theory. It is known to be in the class NP and has been largely studied since the 1970's (Read and Corneil 1977). The notion of graph isomorphism is also directly related to the important notion of invariance to node permutation, as we discuss in Section 4.

3.3 Graph Neural Networks

Consider a graph G with adjacency matrix A and feature matrix X. Graph Neural Networks (GNNs) use the graph structure (A) and the node features (X) to learn a node-level or a graph-level representation—or *embedding*—of G. GNNs iteratively update a node representation by aggregating its neighbors' representations. At iteration l, a node representation captures its l-hop neighborhood's structural information. Formally, the lth layer of a general GNN can be defined as follows:

$$\mathrm{a}_i^{l+1} = \text{AGGREGATE}^l(\{\mathrm{z}_j^l : j \in N(i)\}) \tag{1}$$

$$\mathrm{z}_i^{l+1} = \text{COMBINE}^l(\mathrm{z}_i^l, \mathrm{a}_i^{l+1}) \ , \tag{2}$$

where z_i^{l+1} is the feature vector of node v_i at layer l and where $\mathrm{z}_i^0 = \mathrm{x}_i$. While COMBINE usually consists in concatenating node representations from different layers, different—and often complex—architectures for AGGREGATE have been proposed. In (Kipf and Welling 2017), the presented GCN merges the AGGREGATE and COMBINE functions as follows:

$$\mathrm{z}_i^{l+1} = \text{ReLU}\left(\text{mean}(\{\mathrm{z}_j^l : j \in N(i) \cup \{i\}\}) \cdot \mathrm{W}^l\right) \ , \tag{3}$$

where ReLU is a rectified linear unit and W^l is a trainable weight matrix. GNNs for graph classification have an additional module that aggregates the node-level representations to produce a graph-level one as follows:

$$\mathrm{z}_G = \text{READOUT}(\{\mathrm{z}_i^L : v_i \in V\}) \ , \tag{4}$$

for a GNN with L layers. In (Xu et al. 2019), the authors discuss the impact that the choice of AGGREGATE^l, COMBINE^l, and READOUT has on the so-called *expressiveness* of the GNN, that is, its ability to map different graphs to different embeddings. They present theoretical conditions under which a GNN is maximally expressive.

We now present a simple yet practical GNN architecture on which we illustrate our experimental framework.

4 Simple Permutation-Invariant Graph Convolutional Network (SPI-GCN)

We carry out our empirical study on a Simple Permutation-Invariant Graph Convolutional Network (SPI-GCN). SPI-GCN's architecture consists of the following sequential modules: 1) a *graph convolution module* that encodes local graph structure and node features in a substructure feature matrix whose rows represent the nodes of the graph, 2) a *sum-pooling layer* as a READOUT function to produce a single-vector representation of the input graph, and 3) a *prediction module* consisting of dense layers that reads the vector representation of the graph and outputs predictions. Figure 1 in the Supplementary Material summarizes the general architecture of SPI-GCN.

Let G be a graph represented by the adjacency matrix $\mathrm{A} \in \mathbb{R}^{n \times n}$ and the feature matrix $\mathrm{X} \in \mathbb{R}^{n \times d}$, where n and d represent the number of nodes and the dimension of the feature space respectively. Without loss of generality, we consider graphs without self-loops, i.e. the adjacency matrix A has zeros on its diagonal. Additionally, when node features are not available (purely structural graphs), we take $\mathrm{X} = \mathrm{I}_n$, where $\mathrm{I}_n \in \mathbb{R}^{n \times n}$ is the identity matrix.

4.1 Graph Convolution Module

Given a graph G with its adjacency and feature matrices, A and X, we define the first convolution layer as follows:

$$\mathrm{Z} = f(\hat{\mathrm{D}}^{-1}\hat{\mathrm{A}}\mathrm{X}\mathrm{W}) \ , \tag{5}$$

where $\hat{\mathrm{A}} = \mathrm{A} + \mathrm{I}_n$ is the adjacency matrix of G with added self-loops, $\hat{\mathrm{D}}$ is the diagonal node degree matrix of $\hat{\mathrm{A}}$,[1] $\mathrm{W} \in \mathbb{R}^{d \times d'}$ is a trainable weight matrix, f is a nonlinear activation function, and $\mathrm{Z} \in \mathbb{R}^{n \times d'}$ is the convolved graph.

The convolution operator (5) computes new node representations as follows. For each node i,[2] the average of its feature vector and the feature vectors of its neighbors is computed and stored in the ith row of the matrix $\hat{\mathrm{X}} := \hat{\mathrm{D}}^{-1}\hat{\mathrm{A}}\mathrm{X} \in \mathbb{R}^{n \times d}$. $\hat{\mathrm{X}}$ is then mapped to a new d'-dimensional feature space through multiplication by W. Finally, a nonlinear activation function f is applied element-wise resulting in a $n \times d'$ substructure feature matrix Z that contains the convolution result. To stack multiple convolution layers, we generalize the propagation rule in (5) as follows:

$$\mathrm{Z}^{l+1} = f^l(\hat{\mathrm{D}}^{-1}\hat{\mathrm{A}}\mathrm{Z}^l\mathrm{W}^l) \ , \tag{6}$$

where $\mathrm{Z}^0 = \mathrm{X}$, Z^l is the output of the lth convolution layer, W^l is a trainable weight matrix, and f^l is the nonlinear activation function applied at layer l. Each row of the resulting matrix Z^{l+1} contains a node representation in a new feature space. Similarly to the GCN presented in (Kipf and Welling 2017) from which we draw inspiration, our graph convolution module merges the AGGREGATE and COMBINE functions (see (1) and (2)), and we can rewrite (6) as:

$$\mathrm{z}_i^{l+1} = f^l\left(\text{mean}(\{z_j^l : j \in N(i) \cup \{i\}\}) \cdot \mathrm{W}^l\right) \ , \tag{7}$$

where z_i^{t+1} is the ith row of Z^{l+1}.

We return the result of the last convolution layer, that is, for a network with L convolution layers, the result of the convolution is the last substructure feature matrix Z^L. Note that (6) accepts graphs with varying node numbers without

[1] If G is a directed graph, $\hat{\mathrm{D}}$ corresponds to the *outdgree* diagonal matrix of $\hat{\mathrm{A}}$.

[2] For convenience, we use i instead of v_i to denote the ith node of a graph.

changing the structure of the convolution layer, i.e. using the same weight matrix W^l.

Our graph convolution model is connected to the 1-dimensional Weisfeiler-Lehman (WL) algorithm (Weisfeiler and Lehman 1968) as shown in (Kipf and Welling 2017; Zhang et al. 2018). The WL algorithm iteratively computes a vertex coloring for a given graph and is applied in practice to the GI problem. The output of the convolution layer in (6) can be viewed as the vertex coloring computed by the 1-dimensional WL algorithm. This parallel with the WL algorithm allows to define invariant pooling operators such as the SortPooling layer presented in (Zhang et al. 2018) and our sum-pooling layer that we define next.

4.2 Sum-Pooling Layer

The *sum-pooling* layer produces a graph-level representation z_G by summing the rows of the node-level representation Z^L returned by the convolution module. Formally:

$$\mathrm{z}_G = \sum_{i=1}^{n} \mathrm{z}_i^L \ . \tag{8}$$

The resulting vector $\mathrm{z}_G \in \mathbb{R}^{d_L}$ contains the final vector representation (or *embedding*) of the input graph G in a d_L-dimensional space. This vector representation is then used for prediction—graph classification in our case.

Using the sum-pooling layer in (8) results in the invariance of our architecture to node permutation as stated in Theorem 1. This invariance property is crucial for GNNs, as it ensures that two isomorphic—and hence equivalent—graphs will result in the same output.

Theorem 1. *Let G and G_ς be two arbitrary isomorphic graphs. The sum-pooling layer of SPI-GCN produces the same vector representation for G and G_ς.*

The proof of Theorem 1 is straightforward and can be found in the Supplementary Material.

Using a summing-based pooling operator (e.g. sum or average of node features) is a simple idea that has already been implemented in graph neural networks such as (Atwood and Towsley 2016; Simonovsky and Komodakis 2017). The key advantage of summing-based methods is their efficiency and inherent invariance to node permutation. Their main drawback, on the other hand, is that by summing node features, we lose more refined information on individual nodes and on the global structure of the graph. We show through our work, however, that summing-based architectures are competitive with more complex deep learning graph architectures.

4.3 Prediction Module

The prediction module of SPI-GCN is a simple MLP that takes as input the graph-level representation z_G returned by the sum-pooling layer and returns either: (i) a probability p in case of binary classification or (ii) a vector p of probabilities such that $\sum_i p_i = 1$ in case of multi-class classification.

Note that SPI-GCN can be trained in an end-to-end fashion through backpropagation. Additionally, since only one graph is treated in a forward pass, the training complexity of SPI-GCN is linear in the number of graphs.

In the next section, we describe a practical methodology for studying the expressiveness of SPI-GCN and its connection to the generalization performance of the algorithm.

5 Investigating Expressiveness of SPI-GCN

We start here by introducing a practical definition of expressiveness. We then show how the defined measure can be used to train SPI-GCN and help understand the impact expressiveness has on its generalization performance.

5.1 Practical Measure of Expressiveness

The expressiveness of a GNN, as defined in (Xu et al. 2019), is its ability to map different graph structures to different embeddings and, therefore, reflects the injectivity of its graph embedding function. Since studying injectivity can be tedious, we characterize expressiveness—and hence injectivity—as a function of the pairwise distance between graph embeddings.

Let $\{\mathrm{z}_{G_i}\}_{i=1}^m$ be the set of graph embeddings computed by a GNN $\mathcal{A}$ for a given input graph data set $\{G_i\}_{i=1}^m$. We define $\mathcal{A}$'s expressiveness, $\mathcal{E}(\mathcal{A})$, as follows:

$$\mathcal{E}(\mathcal{A}) = \text{mean}(\{||\mathrm{z}_{G_i} - \mathrm{z}_{G_j}||_2 : i, j = 1, \ldots, m\}) \ , \tag{9}$$

that is, the average pairwise Euclidean distance between graph embeddings produced by $\mathcal{A}$. While not strictly equivalent to injectivity, $\mathcal{E}$ is a reasonable indicator thereof, as the average pairwise distance reflects the *diversity* within graph representations which, in turn, is expected to be higher for more diverse input graph data sets. For permutation-invariant GNNs like SPI-GCN,[3] $\mathcal{E}$ is zero when all graphs $\{G_i\}_{i=1}^m$ are isomorphic.

5.2 Penalized Cross Entropy Loss

We train SPI-GCN using a *penalized cross entropy loss*, $\mathcal{L}_p$, that consists of a classical cross entropy augmented with a penalty term defined as a function of the expressiveness of SPI-GCN. Formally:

$$\begin{aligned} \mathcal{L}_p = \ & \text{cross-entropy}(\{y_i\}_{i=1}^m, \{\hat{y}_i\}_{i=1}^m) \\ & \underbrace{-\alpha \cdot \mathcal{E}(\text{SPI-GCN})}_{\pi} \ , \end{aligned} \tag{10}$$

where $\{y_i\}_{i=1}^m$ (resp. $\{\hat{y}_i\}_{i=1}^m$) is the set of real (resp. predicted) graph labels, α is a non-negative penalty factor, and $\mathcal{E}$ is defined as in (9) with $\{\mathrm{z}_{G_i}\}_{i=1}^m$ being the graph embeddings computed by SPI-GCN.

By adding the penalty term π in $\mathcal{L}_p$, the expressiveness $\mathcal{E}(\text{SPI-GCN})$ is maximized while the cross entropy is minimized during the training process. The penalty factor α controls the importance attributed to $\mathcal{E}(\text{SPI-GCN})$ when $\mathcal{L}_p$ is minimized. Consequently, higher values of α allow to train more expressive variants of SPI-GCN whereas for $\alpha = 0$, only the cross entropy is minimized.

In the next section, we assess the performance of SPI-GCN for different values of α. We also compare SPI-GCN with other more complex approaches, including two graph deep learning methods and one state-of-the-art graph kernel.

[3] As mentioned previously, we state that permutation-invariance is a minimal requirement for any practical GNN.

6 Experiments

We carry out a first set of experiments where we compare our approach, SPI-GCN, with one state-of-the-art graph kernel and two recent deep learning approaches for graph-structured data. In a second set of experiments, we train different instances of SPI-GCN with increasing values of the penalty factor α (see (10)) in an attempt to understand how the expressiveness—or injectivity—of SPI-GCN affects its test accuracy, and whether it is the determining factor of its generalization performance, as implicitly suggested in (Xu et al. 2019). We implement SPI-GCN using PyTorch (Paszke et al. 2017).

6.1 Data Sets

We use nine public benchmark data sets to evaluate the accuracy of SPI-GCN. These data sets include five bioinformatics data sets (MUTAG (Debnath et al. 1991), PTC (Srinivasan et al. 2001), ENZYMES (Borgwardt et al. 2005), NCI1 (Wale, Watson, and Karypis 2008), and PROTEINS (Dobson and Doig 2003)), two social network data sets (IMDB-BINARY and IMDB-MULTI (Yanardag and Vishwanathan 2015)), one image data set where images are represented as region adjacency graphs (COIL-RAG (Riesen and Bunke 2008)), and one synthetic data set (SYNTHIE (Morris et al. 2016)). These data sets are available at (Kersting et al. 2016) in a specific text format that we process in order to transform the graphs into a (adjacency matrix, feature matrix) format that can be processed by our neural network. We also evaluate SPI-GCN on an original real-world data set, HYDRIDES, that contains metal hydrides in a graph representation, labelled as *stable* or *unstable* according to a specific energetic property that determines their ability to store hydrogen efficiently. A detailed description of the HYDRIDES data set is provided in the Supplementary Material.

The properties of the tested data sets are summarized in Table 1 in the Supplementary Material.

6.2 Architecture of SPI-GCN

The instance of SPI-GCN that we use for experiments has two graph convolution layers of 128 and 32 hidden units respectively, followed by a hyperbolic tangent function and a softmax function (per node) respectively. The sum-pooling layer is a classical sum applied row-wise; it is followed by a prediction module consisting of a MLP with one hidden layer of 256 hidden units followed by a batch normalization layer and a ReLU. We choose this architecture by trial and error and keep it unchanged throughout the experiments.

6.3 Comparison with Other Methods

In these experiments, we consider the simplest variant of SPI-GCN where the penalty term π in (10) is discarded by setting $\alpha = 0$. That is, the algorithm is trained using only the cross entropy loss.

Baselines We compare SPI-GCN with the state-of-the-art Weisfeiler-Lehman subtree kernel (WL) (Shervashidze et al. 2011), the well-known graph neural network PATCHY-SAN (PSCN) (Niepert, Ahmed, and Kutzkov 2016), and the more recent deep learning approach Deep Graph Convolutional Neural Network (DGCNN) (Zhang et al. 2018) that uses a similar convolution module to ours.

Experimental procedure We train SPI-GCN using full batch ADAM optimizer (Kingma and Ba 2015), with cross entropy as the loss function to minimize ($\alpha = 0$ in (10)). After trying few combination of values, we set ADAM's hyperparameters as follows. The algorithm is trained for 200 epochs on all data sets and the learning rate is set to 10^{-3}. To estimate the accuracy, we perform 10-fold cross validation using 9 folds for training and one fold for testing each time. We report the average (test) accuracy and the corresponding standard deviation in Table 1. Note that we only use node attributes in our experiments. In particular, SPI-GCN does not exploit node or edge labels of the data sets. When node attributes are not available, we use the identity matrix as the feature matrix ($\mathrm{X} = \mathrm{I}_n$) for each graph.

We follow the same procedure for DGCNN. We use the authors' PyTorch implementation[4] and perform 10-fold cross validation with the recommended values for training epochs, learning rate, and the SortPooling parameter k, for each data set (see Table 2 in the Supplementary Material).

For PSCN, we report the results from the original paper (Niepert, Ahmed, and Kutzkov 2016) (for receptive field size $k = 10$) as we could not find an authors' public implementation of the algorithm. The experiments were conducted using a similar procedure as ours.

For WL, we follow (Niepert, Ahmed, and Kutzkov 2016; Yanardag and Vishwanathan 2015) and set the height parameter h to 2. We set the algorithm to use node labels and choose the regularization parameter C of the SVM from $\{10^{-7}, 10^{-5}, \ldots, 10^{7}\}$ using cross validation as follows: we split the data set into a training set (90% of the graphs) and a test set (remaining 10%), then perform 10-fold cross validation on the training set with LIBSVM (Chang and Lin 2011). The parameter C with the highest average validation accuracy is then evaluated on the test set. The experiment is repeated 10 times and we report the average test accuracy and the standard deviation. We use the authors' MATLAB implementation,[5] where we modify the cross validation script to implement the evaluation procedure described previously.[6]

Results Table 1 shows the results for our algorithm (SPI-GCN), DGCNN (Zhang et al. 2018), PSCN (Niepert, Ahmed, and Kutzkov 2016), and WL (Shervashidze et al. 2011). We observe that SPI-GCN is highly competitive with other algorithms despite using the same architecture for all data sets. The only noticeable exceptions are on the NCI1 and IMDB-BINARY data sets, where the best approach (WL) is up to 1.24 times better. On the other hand, SPI-GCN appears to be highly competitive on classification tasks with more than 3 classes (ENZYMES, COIL-RAG, SYNTHIE). The differ-

[4] https://github.com/muhanzhang/pytorch_DGCNN

[5] https://www.bsse.ethz.ch/mlcb/research/machine-learning/graph-kernels/weisfeiler-lehman-graph-kernels.html

[6] The original script returns the average test accuracy of the best C parameters, i.e. parameters with the best validation accuracy on each fold, for one complete run of 10-fold cross validation.

ence in accuracy is particularly significant on COIL-RAG (100 classes), where SPI-GCN is around 34 times better than DGCNN, suggesting that the features extracted by SPI-GCN are more suitable to characterize the graphs at hand. Results for WL on COIL-RAG and SYNTHIE are not available as we could not find these data sets in the appropriate format for the algorithm online. SPI-GCN also achieves a very reasonable accuracy on the HYDRIDES data set.

We expect the accuracy (respectively variance) of SPI-GCN to improve (respectively decrease) after tuning its hyperparameters to individual data sets. SPI-GCN may also benefit from the exploitation of node labels (as additional features) and edge labels (as weights in the adjacency matrix), especially on data sets where it lags behind other approaches, such as NCI1 and IMDB-BINARY.

The results in Table 1 show that despite its simplicity, SPI-GCN is competitive with other practical graph algorithms and, hence, it is a reasonable architecture to consider for our next set of experiments involving expressiveness.

6.4 Expressiveness Experiments

We perform this set of experiments in an attempt to answer the following questions:

- Do more expressive GNNs perform better on graph classification tasks? That is, is the injectivity of a GNN's graph function the determining factor of its performance?
- Can the performance be explained by another factor? If yes, what is it?

To this end, we train increasingly injective instances of SPI-GCN on the penalized cross entropy loss $\mathcal{L}_p$ (10) by setting the penalty factor α to increasingly large values. Then, for each trained instance, we investigate (i) its test accuracy, (ii) its expressiveness $\mathcal{E}(\text{SPI-GCN})$ (9), that is, the average pairwise distance between the graph representations computed by SPI-GCN, and (iii) the *average normalized Inter-class Graph Embedding Distance* (IGED), defined as the average pairwise Euclidean distance between mean graph embeddings taken class-wise divided by $\mathcal{E}(\text{SPI-GCN})$. Formally:

$$\text{IGED} = \frac{\text{mean}(\{||\mathrm{z}_c^* - \mathrm{z}_{c'}^*||_2 : c, c' = 1, \ldots, C\})}{\mathcal{E}(\text{SPI-GCN})} \quad , \quad (11)$$

where z_c^* is the mean graph embedding for class c. The IGED can be interpreted as an estimate of how well the graph embeddings computed by SPI-GCN are *separated* with respect to their respective class.

Experimental procedure We train SPI-GCN on the penalized cross entropy loss $\mathcal{L}_p$ (10) where we sequentially choose α from $\{0, 10^{-3}, 10^{-1}, 1, 10\}$. We do so using full batch ADAM optimizer that we run for 200 epochs with a learning rate of 10^{-3}, on all the graph data sets introduced previously. For each data set and for each value of α, we perform 10-fold cross validation using 9 folds for training and one fold for testing. We report in Table 2 the average and standard deviation of: (a) the test accuracy, (b) the expressiveness $\mathcal{E}(\text{SPI-GCN})$, and (c) the IGED (11), for each value of α and for each data set.

Results We observe from Table 2 that using a penalty term in $\mathcal{L}_p$ to maximize the expressiveness—or injectivity—of SPI-GCN helps to improve the test accuracy on some data sets, notably on MUTAG, PTC, and SYNTHIE. However, larger values of $\mathcal{E}(\text{SPI-GCN})$ do not correspond to a higher test accuracy except for two cases (PTC, SYNTHIE). Overall, $\mathcal{E}(\text{SPI-GCN})$ increases when α increases, as expected, since the expressiveness is maximized during training when $\alpha > 0$. The IGED, on the other hand, is correlated to the best performance in four out of ten cases (ENZYMES, IMDB-BINARY, and IMDB-MULTI), where the test accuracy is maximal when the IGED is maximal. On HYDRIDES, the difference in IGED for $\alpha = 10^{-1}$ (highest accuracy) and $\alpha = 1$ (highest IGED value) is negligible.

Our empirical results indicate that while optimizing the expressiveness of SPI-GCN may result in a higher test accuracy in some cases, more expressive GNNs do not systematically perform better in practice. The IGED, however, which reflects a GNN's ability to compute graph representations that are correctly clustered according to their effective class, better explains the generalization performance of the GNN.

7 Conclusion

In this paper, we challenged the common belief that more expressive GNNs achieve a better performance. We introduced a principled experimental procedure to analyze the link between the expressiveness of a GNN and its test accuracy in a graph classification setting. To the best of our knowledge, our work is the first that explicitly studies the generalization performance of GNNs by trying to uncover the factors that control it, and paves the way for more theoretical analyses. Interesting directions for future work include the design of better expressiveness estimators, as well as different (possibly more complex) penalized loss functions.

References

Atwood, J., and Towsley, D. 2016. Diffusion-convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2001–2009.

Bojchevski, A.; Shchur, O.; Zügner, D.; and Günnemann, S. 2018. NetGAN: Generating graphs via random walks. In *Proceedings of the International Conference on Machine Learning*, volume 80, 609–618.

Borgwardt, K. M.; Ong, C. S.; Schönauer, S.; Vishwanathan, S. V. N.; Smola, A. J.; and Kriegel, H.-P. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21(1):47–56.

Chang, C.-C., and Lin, C.-J. 2011. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2(3):1–27.

Chen, T.; Bian, S.; and Sun, Y. 2019. Are powerful graph neural nets necessary? A dissection on graph classification. arXiv:1905.04579v2.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural language pro-

Table 1: Accuracy results for SPI-GCN, two deep learning methods (DGCNN, PSCN), and a graph kernel method (WL).

Algorithm	SPI-GCN	DGCNN	PSCN	WL
MUTAG	84.40 ± 8.14	86.11 ± 7.14	$\mathbf{88.95 \pm 4.37}$	82.77 ± 8.46
PTC	56.41 ± 5.71	55.00 ± 5.10	$\mathbf{62.29 \pm 5.68}$	57.05 ± 7.61
NCI1	64.11 ± 2.37	72.73 ± 1.56	76.34 ± 1.68	$\mathbf{79.87 \pm 1.77}$
PROTEINS	72.06 ± 3.18	72.79 ± 3.58	$\mathbf{75.00 \pm 2.51}$	72.25 ± 3.22
ENZYMES	50.17 ± 5.60	47.00 ± 8.36	–	$\mathbf{51.16 \pm 5.33}$
IMDB-BINARY	60.40 ± 4.15	68.60 ± 5.66	71.00 ± 2.29	$\mathbf{72.10 \pm 5.30}$
IMDB-MULTI	44.13 ± 4.61	45.20 ± 3.75	45.23 ± 2.84	$\mathbf{51.26 \pm 4.31}$
COIL-RAG	$\mathbf{74.38 \pm 2.42}$	2.21 ± 0.33	–	–
SYNTHIE	$\mathbf{71.00 \pm 6.44}$	54.25 ± 4.34	–	–
HYDRIDES	82.75 ± 2.67	–	–	–

Table 2: Expressiveness experiments results. SPI-GCN is trained on the penalized cross entropy loss, $\mathcal{L}_p$, with increasing values of the penalty factor α. For each data set, and for each value of α, we report the test accuracy (a), the expressiveness $\mathcal{E}$(SPI-GCN) (b), and the IGED (c). Highlighted are the maximal values for each quantity.

α	0	10^{-3}	10^{-1}	1	10	
MUTAG	84.40 ± 8.14	84.40 ± 8.14	$\mathbf{86.07 \pm 9.03}$	82.56 ± 7.33	81.45 ± 6.68	(a)
	0.09 ± 0.01	0.09 ± 0.01	0.12 ± 0.01	5.96 ± 1.08	$\mathbf{6.32 \pm 0.76}$	(b)
	0.68 ± 0.16	0.68 ± 0.16	0.82 ± 0.18	$\mathbf{1.21 \pm 0.23}$	$\mathbf{1.20 \pm 0.22}$	(c)
PTC	56.41 ± 5.71	54.97 ± 6.05	54.64 ± 6.33	57.88 ± 8.65	$\mathbf{58.70 \pm 7.40}$	(a)
	0.09 ± 0.01	0.09 ± 0.01	0.11 ± 0.01	8.41 ± 3.13	$\mathbf{9.03 \pm 2.94}$	(b)
	0.26 ± 0.05	0.26 ± 0.05	0.26 ± 0.06	0.41 ± 0.22	$\mathbf{0.42 \pm 0.22}$	(c)
NCI1	64.11 ± 2.37	$\mathbf{64.21 \pm 2.36}$	64.01 ± 2.87	63.48 ± 1.36	63.19 ± 1.72	(a)
	0.09 ± 0.004	0.09 ± 0.005	1.07 ± 0.19	16.83 ± 0.49	$\mathbf{16.91 \pm 0.52}$	(b)
	0.18 ± 0.02	0.19 ± 0.03	0.59 ± 0.05	$\mathbf{0.62 \pm 0.05}$	$\mathbf{0.62 \pm 0.05}$	(c)
PROTEINS	$\mathbf{72.06 \pm 3.18}$	71.78 ± 3.55	71.51 ± 3.26	70.97 ± 3.49	71.42 ± 3.23	(a)
	5.89 ± 1.34	13.07 ± 3.21	$\mathbf{35.88 \pm 4.89}$	$\mathbf{35.88 \pm 4.89}$	$\mathbf{35.88 \pm 4.89}$	(b)
	$\mathbf{0.74 \pm 0.09}$	$\mathbf{0.74 \pm 0.09}$	$\mathbf{0.74 \pm 0.09}$	$\mathbf{0.74 \pm 0.09}$	$\mathbf{0.74 \pm 0.09}$	(c)
ENZYMES	$\mathbf{50.17 \pm 5.60}$	$\mathbf{50.17 \pm 5.60}$	29.33 ± 5.93	29.33 ± 5.54	29.33 ± 5.88	(a)
	0.79 ± 0.21	1.85 ± 0.64	23.22 ± 2.99	23.33 ± 3.02	$\mathbf{23.35 \pm 3.01}$	(b)
	$\mathbf{0.44 \pm 0.06}$	0.42 ± 0.10	0.42 ± 0.10	0.42 ± 0.10	0.42 ± 0.10	(c)
IMDB-BINARY	60.40 ± 4.15	$\mathbf{61.70 \pm 4.96}$	61.10 ± 3.75	54.40 ± 3.10	54.20 ± 5.15	(a)
	0.12 ± 0.01	0.12 ± 0.01	0.16 ± 0.01	$\mathbf{12.43 \pm 2.37}$	11.70 ± 2.89	(b)
	$\mathbf{0.15 \pm 0.03}$	$\mathbf{0.15 \pm 0.03}$	$\mathbf{0.15 \pm 0.03}$	0.12 ± 0.08	0.12 ± 0.08	(c)
IMDB-MULTI	44.13 ± 4.61	44.60 ± 5.41	$\mathbf{44.80 \pm 4.51}$	39.73 ± 4.34	38.87 ± 4.42	(a)
	0.08 ± 0.01	0.08 ± 0.01	0.64 ± 0.14	$\mathbf{10.38 \pm 1.05}$	9.91 ± 1.15	(b)
	$\mathbf{0.16 \pm 0.02}$	$\mathbf{0.16 \pm 0.02}$	$\mathbf{0.16 \pm 0.09}$	0.15 ± 0.09	0.15 ± 0.09	(c)
COIL-RAG	$\mathbf{74.38 \pm 2.42}$	$\mathbf{74.38 \pm 2.45}$	72.49 ± 3.21	52.08 ± 4.89	28.72 ± 3.62	(a)
	0.08 ± 0.002	0.081 ± 0.002	0.13 ± 0.01	2.00 ± 0.18	$\mathbf{2.33 \pm 0.14}$	(b)
	0.95 ± 0.01	0.95 ± 0.01	0.96 ± 0.01	$\mathbf{0.98 \pm 0.02}$	$\mathbf{0.98 \pm 0.02}$	(c)
SYNTHIE	71.00 ± 6.44	71.00 ± 6.04	$\mathbf{74.00 \pm 6.44}$	73.00 ± 7.57	73.75 ± 7.52	(a)
	1.60 ± 0.20	1.86 ± 0.24	$\mathbf{29.97 \pm 2.16}$	29.50 ± 2.18	29.37 ± 2.18	(b)
	$\mathbf{0.73 \pm 0.07}$	0.72 ± 0.08	0.61 ± 0.11	0.59 ± 0.12	0.58 ± 0.12	(c)
HYDRIDES	82.75 ± 2.67	82.65 ± 2.44	$\mathbf{83.92 \pm 4.30}$	77.45 ± 3.25	76.37 ± 2.57	(a)
	0.13 ± 0.01	0.13 ± 0.01	1.68 ± 0.87	4.75 ± 0.41	$\mathbf{5.03 \pm 0.75}$	(b)
	0.50 ± 0.11	0.50 ± 0.11	0.8 ± 0.19	$\mathbf{0.85 \pm 0.21}$	0.72 ± 0.22	(c)

cessing (almost) from scratch. *Journal of Machine Learning Research* 12:2493–2537.

Debnath, A. K.; Lopez de Compadre, R. L.; Debnath, G.; Shusterman, A. J.; and Hansch, C. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry* 34(2):786–797.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 3844–3852.

Dobson, P. D., and Doig, A. J. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology* 330(4):771–783.

Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, volume 28, 2224–2232.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 1263–1272.

Gori, M.; Monfardini, G.; and Scarselli, F. 2005. A new model for learning in graph domains. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, 729–734.

Haussler, D. 1999. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California, Santa Cruz.

Kersting, K.; Kriege, N. M.; Morris, C.; Mutzel, P.; and Neumann, M. 2016. Benchmark data sets for graph kernels.

Kingma, D. P., and Ba, J. 2015. ADAM: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Kriege, N. M.; Johansson, F. D.; and Morris, C. 2019. A survey on graph kernels. arXiv:1903.11835.

Mckay, B. D., and Piperno, A. 2014. Practical graph isomorphism, ii. *Journal of Symbolic Computation* 60:94–112.

Min, S.; Lee, B.; ; and Yoon, S. 2017. Deep learning in bioinformatics. *Briefings in Bioinformatics* 18(5):851–869.

Morris, C.; Kriege, N. M.; Kersting, K.; and Mutzel, P. 2016. Faster kernels for graphs with continuous attributes via hashing. In *IEEE 16th International Conference on Data Mining*, 1095–1100.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, volume 48, 2014–2023. JMLR.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Read, R. C., and Corneil, D. G. 1977. The graph isomorphism disease. *Journal of Graph Theory* 1(4):339–363.

Riesen, K., and Bunke, H. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, 287–297. Springer Berlin Heidelberg.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The graph neural network model. *Transactions on Neural Networks* 20(1):61–80.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; van den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *The Semantic Web (ESWC)*, 593–607.

Shervashidze, N.; Schweitzer, P.; van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12:2539–2561.

Simonovsky, M., and Komodakis, N. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, 29–38.

Srinivasan, A.; Helma, C.; King, R. D.; and Kramer, S. 2001. The predictive toxicology challenge 2000–2001. *Bioinformatics* 17(1):107–108.

Wale, N.; Watson, I. A.; and Karypis, G. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* 14(3):347–375.

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

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? In *International Conference on Learning Representations*.

Yanardag, P., and Vishwanathan, S. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1365–1374. ACM.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, 4438–4445. AAAI Press.