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], and Jean-Claude Crivello[3]

[1] LTCI, Télécom Paris, Institut Polytechnique de Paris
asma.atamna@telecom-paris.fr
[2] NutriOmics, INSERM, Sorbonne University, Paris, France
nataliya.sokolovska@sorbonne-universite.fr
[3] ICMPE (UMR 7182), CNRS, University of Paris-Est, Thiais, France
crivello@icmpe.cnrs.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.

Keywords: Graph neural networks · Classification · Expressiveness.

1 Introduction

Many real-world data present an inherent structure and can be modelled as sequences, graphs, or hypergraphs [5,15,9,2]. 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 and their respective labels $\{y_i\}_{i=1}^m$, where $y_i \in \{1, \ldots, C\}$ and C is the number of classes, we aim at finding a mapping $f_\theta : \mathcal{G} \to \{1, \ldots, C\}$ that minimizes the classification error, where θ denotes the parameters to optimize.

* Supported by the Emergence@INC-2018 program of the French National Center for Scientific Research (CNRS) and the *DiagnoLearn* ANR JCJC project.

Graph neural networks (GNNs) and their deep learning variants, the graph convolutional networks (GCNs) [7,13,1,9,27,17,10,20], have gained considerable interest recently. 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 learns and how the learned graph (or node) mapping influences its generalization performance. In a recent work, [25] 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 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.

To illustrate our experimental framework, we introduce a simple yet practical architecture, the Simple Permutation-Invariant Graph Convolutional Network (SPI-GCN). 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

Graph neural networks (GNNs) were first introduced in [11,19]. They learn latent node representations by iteratively aggregating neighborhood information for each node. Their more recent deep learning variants, the graph convolutional networks (GCNs), generalize conventional convolutional neural networks to irregular graph domains. In [13], 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 Weisfeiler-Lehman (WL) algorithm [24]. A related GCN that is invariant to node permutation is presented in [27]. The graph convolution operator is closely related to the one in [13], 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 GCN is PATCHY-SAN [17]. 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 [14].

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 [25] 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. The authors also present the Graph Isomorphism Network (GIN), which approximates the theoretical maximally expressive GNN. In another study [4], 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 [25] 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 [13] to graph-level classification. Our SPI-GCN is also related to [27] in that we use a similar graph convolution operator inspired by [13]. Unlike [27], however, our architecture does not require any node ordering, and we only use a simple multilayer perceptron (MLP) to perform classification.

3 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 this work, we represent a graph G by two matrices: (i) an *adjacency matrix* $\mathrm{A} \in \mathbb{R}^{n \times n}$ such that $a_{ij} = 1$ if there is an edge between nodes v_i and v_j and $a_{ij} = 0$ otherwise,[4] and (ii) a *node feature matrix* $\mathrm{X} \in \mathbb{R}^{n \times d}$, with d being the number of node features. 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.

An important notion in graph theory is *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.

[4] Given a matrix M, m_i denotes its ith row and m_{ij} denotes the entry at its ith row and jth column. More generally, we denote matrices by capital letters and vectors by small letters. Scalars, on the other hand, are denoted by small italic letters.

Informally, this definition 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.

3.1 Graph Neural Networks

Consider a graph G with adjacency matrix A and feature matrix X. 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:

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

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

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 [13], the presented GCN merges the AGGREGATE and COMBINE functions as follows:

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

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:

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

for a GNN with L layers. In [25], the authors discuss the impact that the choice of $\mathrm{AGGREGATE}^l$, $\mathrm{COMBINE}^l$, 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)

Our Simple Permutation-Invariant Graph Convolutional Network (SPI-GCN) 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.

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 $\hat{\mathrm{A}}$,[5] $\mathrm{W} \in \mathbb{R}^{d \times d'}$ is a trainable weight matrix, f is a nonlinear activation function, and $\mathrm{Z} \in \mathbb{R}^{n \times d'}$ is the convolved graph. 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. Similarly to the GCN presented in [13] 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:

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

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) is able to process graphs with varying node numbers.

4.2 Sum-Pooling Layer

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

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

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

The resulting vector $z_G \in \mathbb{R}^{d_L}$ 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 a sum pooling operator is a simple idea that has been used in GNNs such as [1,21]. Additionally, it results in the invariance of our architecture to node permutation, as stated in Theorem 1.

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.*

This invariance property is crucial for GNNs as it ensures that two isomorphic—and hence equivalent—graphs will result in the same output. The proof of Theorem 1 is straightforward and omitted for space limitations.

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 back-propagation. 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 [25], 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 $\{z_{G_i}\}_{i=1}^m$ 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:

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

that is, $\mathcal{E}(\mathcal{A})$ 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,[6] $\mathcal{E}$ 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:

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

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 in (9) with $\{\mathrm{z}_{G_i}\}_{i=1}^m$ being the graph embeddings computed by SPI-GCN.

By adding the penalty term $-\alpha \cdot \mathcal{E}(\text{SPI-GCN})$ in $\mathcal{L}_p$, the expressiveness 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 GNN architectures, including the state-of-the-art method.

6 Experiments

We carry out a first set of experiments where we compare our approach, SPI-GCN, with two recent GCNs. 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 of SPI-GCN affects its test accuracy, and whether it is the determining factor of its generalization performance, as implicitly suggested in [25]. Our code and data are available at https://github.com/asmaatamna/SPI-GCN.

6.1 Data Sets

We use nine public benchmark data sets including five bioinformatics data sets (MUTAG [6], PTC [22], ENZYMES [3], NCI1 [23], PROTEINS [8]), two social network data sets (IMDB-BINARY, IMDB-MULTI [26]), one image data set where images are represented as region adjacency graphs (COIL-RAG [18]), and

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

one synthetic data set (SYNTHIE [16]). We also evaluate SPI-GCN on an original real-world data set collected at the ICMPE,[7] HYDRIDES, that contains metal hydrides in graph format, labelled as *stable* or *unstable* according to specific energetic properties that determine their ability to store hydrogen efficiently.

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 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 well-known GCN, PATCHY-SAN (PSCN) [17], the Deep Graph Convolutional Neural Network (DGCNN) [27] that uses a similar convolution module to ours, and the recent state-of-the-art Graph Isomorphism Network (GIN) [25].

Experimental procedure. We train SPI-GCN using full batch ADAM optimizer [12], with cross entropy as the loss function to minimize ($\alpha = 0$ in (10)). Upon experimentation, 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 for each graph.

We follow the same procedure for DGCNN. We use the authors' implementation[8] and perform 10-fold cross validation with the recommended values for training epochs, learning rate, and SortPooling parameter k, for each data set.

For PSCN, we report the results from the original paper [17] (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 GIN, we also report the published results [25] (GIN-0 in the paper), as it was not straightforward to use the authors' implementation.

[7] East Paris Institute of Chemistry and Materials Science, France.
[8] https://github.com/muhanzhang/pytorch_DGCNN

Results. Table 1 shows the results for our algorithm (SPI-GCN), DGCNN [27], PSCN [17], and the state-of-the-art GIN [25]. 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 (GIN) is up to 1.28 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 difference 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. SPI-GCN also achieves a very reasonable accuracy on the HYDRIDES data set and is 1.06 times better than DGCNN on ENZYMES.

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.

Table 1. Accuracy results for SPI-GCN and three other deep learning methods (DGCNN, PSCN, GIN).

Algorithm	SPI-GCN	DGCNN	PSCN	GIN
MUTAG	84.40 ± 8.14	86.11 ± 7.14	88.95 ± 4.37	$\mathbf{89.4 \pm 5.6}$
PTC	56.41 ± 5.71	55.00 ± 5.10	62.29 ± 5.68	$\mathbf{64.6 \pm 7.0}$
NCI1	64.11 ± 2.37	72.73 ± 1.56	76.34 ± 1.68	$\mathbf{82.7 \pm 1.7}$
PROTEINS	72.06 ± 3.18	72.79 ± 3.58	75.00 ± 2.51	$\mathbf{76.2 \pm 2.8}$
ENZYMES	$\mathbf{50.17 \pm 5.60}$	47.00 ± 8.36	–	–
IMDB-BINARY	60.40 ± 4.15	68.60 ± 5.66	71.00 ± 2.29	$\mathbf{75.1 \pm 5.1}$
IMDB-MULTI	44.13 ± 4.61	45.20 ± 3.75	45.23 ± 2.84	$\mathbf{52.3 \pm 2.8}$
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	–	–	–

6.4 Expressiveness Experiments

Through these experiments, we try 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 $\mathcal{E}$(SPI-GCN) (9), 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 $\mathcal{E}$(SPI-GCN). Formally:

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

where z_k^* is the mean graph embedding for class k. 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 $\mathcal{E}$(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 $\mathcal{E}$(SPI-GCN) do not correspond to a higher test accuracy except for two cases (PTC, SYNTHIE). Overall, $\mathcal{E}$(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

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-BIN.	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-MUL.	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)

better expressiveness estimators, as well as different (possibly more complex) penalized loss functions.

References

1. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: NIPS (2016)
2. Bojchevski, A., Shchur, O., Zügner, D., Günnemann, S.: NetGAN: Generating graphs via random walks. In: ICML (2018)
3. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S.V.N., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(1), 47–56 (2005)
4. Chen, T., Bian, S., Sun, Y.: Are powerful graph neural nets necessary? A dissection on graph classification (2019), arXiv:1905.04579v2
5. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. Journal of Machine Learning Research **12**, 2493–2537 (2011)

6. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: 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 (1991)
7. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: NIPS (2016)
8. Dobson, P.D., Doig, A.J.: Distinguishing enzyme structures from non-enzymes without alignments. Journal of Molecular Biology **330**(4), 771–783 (2003)
9. Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. In: NIPS (2015)
10. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: ICML (2017)
11. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: IJCNN (2005)
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
13. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
14. Mckay, B.D., Piperno, A.: Practical graph isomorphism, ii. Journal of Symbolic Computation **60**, 94–112 (2014)
15. Min, S., Lee, B., , Yoon, S.: Deep learning in bioinformatics. Briefings in Bioinformatics **18**(5), 851–869 (2017)
16. Morris, C., Kriege, N.M., Kersting, K., Mutzel, P.: Faster kernels for graphs with continuous attributes via hashing. In: ICDM (2016)
17. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: ICML (2016)
18. Riesen, K., Bunke, H.: 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. pp. 287–297. Springer Berlin Heidelberg (2008)
19. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. Transactions on Neural Networks **20**(1), 61–80 (2009)
20. Schlichtkrull, M., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: The Semantic Web (ESWC) (2018)
21. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: CVPR 2017 (2017)
22. Srinivasan, A., Helma, C., King, R.D., Kramer, S.: The predictive toxicology challenge 2000–2001. Bioinformatics **17**(1), 107–108 (2001)
23. Wale, N., Watson, I.A., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. Knowledge and Information Systems **14**(3), 347–375 (2008)
24. Weisfeiler, B., Lehman, A.A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia **9**(2), 12–16 (1968)
25. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019)
26. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: SIGKDD (2015)
27. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: AAAI (2018)