RuleGNNs: A Rule Based Approach for Learning on Graphs*

Florian Seiffarth$^{[0009-0004-5251-0894]}$

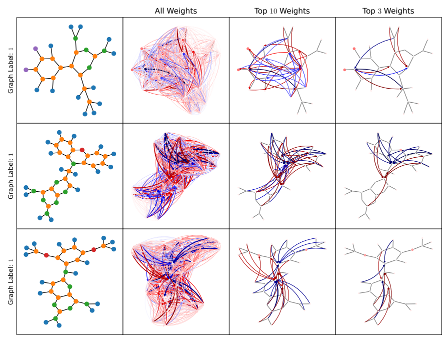
University of Bonn, Friedrich-Hirzebruch-Allee 8, 53113 Bonn, Germany
seiffarth@cs.uni-bonn.de

Abstract. A common problem of classical graph neural networks is their limitation regarding long range dependencies, varying graph sizes, restricted expressive power and their lack of interpretability. To overcome these limitations, we propose a new type of graph neural networks called RuleGNNs. The advantage of RuleGNNs is that their architecture is dynamic and depends on the input graphs. More precisely, their architecture can be designed by formal rules, e.g., using expert knowledge. We prove that RuleGNNs are by definition permutation invariant, i.e., the order of the nodes in the input graph does not affect the output. Moreover, their distinguishing power depends on the chosen rules and can be adapted to the specific problem. Our experiments show that the predictive performance of RuleGNNs is comparable to state-of-the-art graph classification algorithms using simple rules based on Weisfeiler-Leman labeling and pattern counting. In addition, we show that RuleGNNs give interpretable results by visualizing the learned weights and biases. Finally, we introduce new synthetic benchmark graph datasets to outline the strengths of RuleGNNs compared to ordinary graph neural networks.

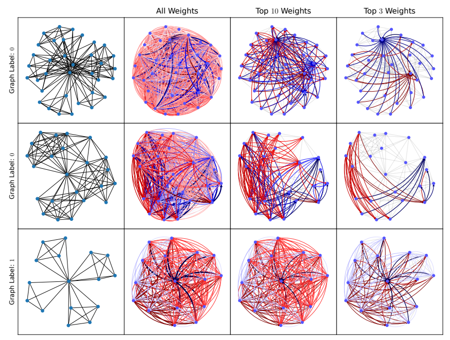
1 Introduction

The message passing paradigm [10] is one of the most successful approaches for classifying graphs. Information is aggregated over the neighborhood of a node and the node embeddings are updated based on this information. Nevertheless, the paradigm has some limitations. Graph neural networks tend to over-smooth the node embeddings, i.e., the embeddings of nodes in a graph become more similar with increasing number of layers [15]. Thus, long range dependencies in a graph are hard to capture. If the graph neural network is based on pure message passing, its distinguishing power is restricted to the 1-WL test [17]. Another problem is the interpretability of the model because of the heterogeneity of the data. By proposing a new type of dynamic neural network layer called rule based layer in this work we aim to overcome these limitations. The main idea is to use formal rules that determine the positions of the weights in the weight matrix and the bias terms. In difference to ordinary network layers we consider a set Θ of learnable parameters instead of fixed weight matrices. We construct the

* The long version of this paper is available at https://arxiv.org/abs/2406.09954



(a) DHFR



(b) IMDB-BINARY

Fig. 1: Visualization of the learned parameters of the best RuleGNN model on DHFR (a) and IMDB-BINARY (b) for three different random graphs from the test set. The label of the graph is given on the left side of the figure. Positive weights are denoted by red arrows and negative weights by blue arrows. The thickness and color corresponds to the absolute value of the weight. The size of the nodes corresponds to the bias values. The second to fourth columns of (a) resp. (b) show all, 10 and the 3 largest positive and negative weights.

weight matrices and bias terms depending on the input sample using learnable parameters from Θ. More precisely, each learnable parameter in Θ is associated with a specific relation between an input and output feature of a layer increasing the interpretability of the model. As an example consider Figure 1 where each input and output feature corresponds to a specific node in the graph. The input samples are (a) molecule graphs resp. (b) snippets of social networks and the task is to predict the graph class. Each colored arrow in the figure corresponds to a parameter from Θ, i.e., a specific relation between two atoms in the molecules or two nodes in the social network. Considering only the 3 parameters with the largest absolute values, see the last column of (a) resp. (b), we see that our approach has learned to propagate information from outer atoms to the rings respectively from the nodes to the “important” nodes of the social network. This example shows several advantages of our approach: (1) the architecture is very *flexible* compared to classical architectures and allows to deal with *arbitrary input dimensions*, (2) messages can pass over *arbitrary distances* in one layer, and (3) the learned parameters and hence also the models are *interpretable* and can be used to extract new knowledge from the data or to improve existing rules.

In this work we introduce RuleGNNs, a new type of graph neural networks, that are based on concatenations of rule based layers. Considering various real-world graph datasets, we demonstrate that RuleGNNs are competitive with state-of-the-art graph neural networks and other graph classification methods. Using synthetic graph datasets we show that “expert knowledge” is easily integrable into our neural network architecture and leads to better classification results.

The rest of the paper is structured as follows. In Section 2 we discuss related work. In Section 3 we introduce the concept of rule based layers and show how they can be used to define RuleGNNs. In Section 4 we present our experiments and results. Finally, in Section 5 we conclude the paper.

2 Related Work

Graph neural networks based on the message passing paradigm [10] have been successfully applied to graph classification [14,11,26,27]. The limitations mentioned in the introduction are addressed by various recent algorithms. Using k-hop approaches aggregating information over long distances has been considered [1,19]. To overcome the limitations of 1-WL test recent algorithms use additional information like subgraph structures or topological information to improve the performance [2,3,4,25] In [30] the authors show that graph neural networks can learn chemical rules like the ortho-para rule for molecules which goes in the direction of interpretability. To increase the interpretability and explainability of graph neural networks there exist different approaches [28,20]. Moreover, also dynamic approaches are considered in the literature [24].

In contrast to these approaches, we provide a simple and general scheme to overcome different limitations of GNNs at once. In fact, we are not aware of any other approach that is able to dynamically adjust the architecture for each input sample based on additional information using formal rules. Moreover, rule based layers are easily integrable into existing architectures and can be extended to other tasks like node classification and even to other domains, e.g., images [22].

3 Rule based layers and RuleGNNs

In this section we first present a new type of neural network layer called ruled based layer that gives rise to a new type of graph neural networks called RuleGNNs. One of the main advantages of RuleGNNs is that their dynamic neural network architecture is freely configurable using almost arbitrary formal rules. We first present the basic notions and the definition of a rule based layer followed by the definition of RuleGNNs. An example based on molecule graphs to illustrate the concept of RuleGNNs is given in Appendix B.

3.1 Preliminaries

For some $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \dots, n\}$. A graph is a pair $G = (V, E)$ with V denoting the set of nodes of G and $E \subseteq \{\{i, j\} \mid i, j \in V\}$ the set of edges. All graphs are undirected and do not contain self-loops or parallel edges. In case that it is clear from the context we omit G and only use V and E. The distance between two nodes $i, j \in V$ in a graph, i.e., the length of the shortest path between i and j, is denoted by $d(i, j)$. A labeled graph is a graph $G = (V, E, l)$ equipped with a function $l : V \to \mathcal{L}$ that assigns to each node a label from the set $\mathcal{L} \subseteq \mathbb{N}$.

In this paper the input samples corresponding to a graph $G = (V, E)$ are always vectors of length equal to $|V|$. In particular, the input vectors can be interpreted as signals over the graph and each dimension of the input vector corresponds to the one-dimensional input signal of a graph node. A rule based graph neural network (RuleGNN) is a function $\mathbf{f}(-, \Theta, \mathcal{R}) : \mathbb{R}^* \longrightarrow \mathbb{R}^*$ depending on a set of learnable parameters Θ and formal rules $\mathcal{R}$. The definition of rules is given in (2). Informally, a rule $\mathbf{R} \in \mathcal{R}$ is a function that determines the distribution of the weights in the weight matrix or the bias term of a layer certain layer of f. The notation $*$ in the domain and codomain of $\mathbf{f}$ indicates that the input and output can be of arbitrary dimension. As usual $\mathbf{f}$ is a concatenation of sub-functions $f^1, \ldots, f^l$ called the layers. More precisely, the i-th layer is a function $f^i(-, \Theta^i, \mathbf{R}^i) : \mathbb{R}^* \longrightarrow \mathbb{R}^*$ where Θ^i is a subset of the learnable parameters Θ and $\mathbf{R}^i$ is an element of the ruleset $\mathcal{R}$. The input data is a triple $(\mathbf{D}, \mathbf{L}, \mathbf{I})$ where $\mathbf{D} = \{x_1 \ldots, x_k\}$ is the set of input samples or input signals with $x_i \in \mathbb{R}^{|V|}$ for the corresponding graph $G = (V, E)$. The labels are denoted by $\mathbf{L} = (y_1 \ldots, y_k)$ with $y_i \in \mathbb{R}^*$ and $\mathbf{I}$ is some additional information known about $\mathbf{D}$, e.g., the graph structure, node or edge labels or the importance of certain neighborhoods. We assume that $\mathbf{I}$ can be used to derive a meaningful set of rules $\mathcal{R}$.

3.2 Rule Based Layer

In this section we will give a formal definition of a rule based layer. Let $f^i(-, \Theta^i, \mathbf{R}^i) : \mathbb{R}^* \longrightarrow \mathbb{R}^*$ be the i-th layer of $\mathbf{f}$. For simplicity, we assume $i = 1$ and omit the indices, i.e., we write $f \coloneqq f^i$, $\Theta \coloneqq \Theta^i$ and $\mathbf{R} \coloneqq \mathbf{R}^i$. Let $x \in \mathbf{D}$ with $x \in \mathbb{R}^n$ be some input signal of a graph $G = (V, E)$ with $|V| = n$. Then $f(-, \Theta, \mathbf{R}) : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ for $n, m \in \mathbb{N}$ is given by

$$f(x, \Theta, \mathbf{R}) = \sigma(W_{\mathbf{R}_W(x)} \cdot x + b_{\mathbf{R}_b(x)}) \ . \tag{1}$$

Here σ denotes an arbitrary activation function and $W_{\mathbf{R}_W(x)} \in \mathbb{R}^{m \times n}$ rsp. $b_{\mathbf{R}_b(x)} \in \mathbb{R}^m$ is some weight matrix rsp. bias term depending on the input vector x and the rule $\mathbf{R}$. The set $\Theta \coloneqq \{w_1, \ldots, w_N, b_1, \ldots, b_M\}$ consists of all possible learnable parameters of the layer. The parameters $\{w_1, \ldots, w_N\}$ are possible entries of the weight matrix while $\{b_1, \ldots, b_M\}$ are possible entries of the bias vector. The key point here is that the rule $\mathbf{R}$ determines the choices and the positions of the weights from Θ in the weight matrix $W_{\mathbf{R}_W(x)}$ and the bias vector $b_{\mathbf{R}_b(x)}$ depending on the input signal x. In particular, *not* all learnable parameters must be used in the weight matrix and the bias vector for some input sample x. In contrast to ordinary neural network layers the weight matrix and the bias vector are not fixed but *functions* of the input signal x. Moreover, for two signals $x, y \in \mathbf{D}$, e.g., $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^k$ with $n \neq k$ corresponding to graphs of different sizes the weight matrices $W_{\mathbf{R}_W(x)}$ and $W_{\mathbf{R}_W(y)}$ also have different dimensions and the learnable parameters can be in totally different positions in the weight matrix. Given the set of learnable parameters $\Theta \coloneqq \{w_1, \ldots, w_N, b_1, \ldots, b_M\}$, for each input signal $x \in \mathbb{R}^n$ each rule $\mathbf{R}$ induces two rule functions

$$\mathbf{R}_W(x) : [m] \times [n] \longrightarrow \{0\} \cup [N] \quad \text{and} \quad \mathbf{R}_b(x) : [m] \longrightarrow \{0\} \cup [M] \tag{2}$$

where $m \in \mathbb{N}$ is the output dimension of the layer that can also depend on x. For simplicity, we assume that matrix and vector indices start at 1 and not at 0. Using the associated rule functions (2) we construct the weight matrix resp. bias vector by defining the entry $(i, j) \in \mathbb{R}^{m \times n}$ in the i-th row and the j-th column of the weight matrix $W_{\mathbf{R}(x)} \in \mathbb{R}^{m \times n}$ via

$$W_{\mathbf{R}_W(x)}(i, j) := \begin{cases} 0 & \text{if } \mathbf{R}_W(x)(i, j) = 0 \\ w_{\mathbf{R}_W(x)(i,j)} & \text{o.w.} \end{cases} \tag{3}$$

and the entry at position k in the bias vector $b_{\mathbf{R}_b(x)} \in \mathbb{R}^m$ by

$$b_{\mathbf{R}_b(x)}(k) := \begin{cases} 0 & \text{if } \mathbf{R}_b(x)(k) = 0 \\ b_{\mathbf{R}_b(x)(k)} & \text{o.w.} \end{cases} . \tag{4}$$

Hence, an entry of the weight matrix or the bias vector is zero if the value of the rule function is zero, otherwise the entry is the learnable parameter from the set Θ and the index is given by the rule function. More precisely, the rule controls the connection between the i-th input and the j-th output feature in the weight matrix. If $m = n$, the multiplication of the weight matrix with the input vector can be interpreted as a special case of signal processing with learnable parameters. Note that in contrast to ordinary message passing the signal processing is not restricted to the neighbors of a node but can be done over the whole graph in one layer only depending on the rule $\mathbf{R}$. The definition in (2) allows arbitrary rule functions. Of course, not all valid functions are useful. In case of graphs only such rule functions are of interest that are equivariant resp. invariant under node permutations as there is no canonical order of graph nodes. Thus, in the following section we will present a construction instruction for an interesting subclass of equivariant rule functions for graphs.

3.3 Graph Rules

Note that for $m = n = |V|$ the rule functions as defined in (2) can be interpreted as a mapping from node pairs $(i, j) \in V \times V$ ($\mathbf{R}_W$) or nodes $i \in V$ ($\mathbf{R}_b$) to 0 or the index of the learnable parameter in Θ. Two different node pairs (i, j) and (k, l) should map to the same integer if and only they "behave similar" in the graph. Our starting point for a general scheme to define rule functions for graphs is a labeling function $l : V \to \mathcal{L}$. In case of molecule graphs take for example the atom labels as node labels, see Appendix B for an example. For unlabeled graphs it is possible to use the degree of a node. Moreover, we use a property function $p : V \times V \to 0 \cup \mathbb{N}$ that defines a relation between two nodes $i, j \in V$ in a graph. Examples are the distance between two nodes, the type of edge connecting the nodes or the information that i and j are in one circle or not. In this way we assign a triple $t(i,j) = (l(i), l(j), p(i,j))$ to each pair of nodes (i,j). Wit this preliminary work we can define $\mathbf{R}_W$ and $\mathbf{R}_b$ from (2) as follows. We recall that the output of $\mathbf{R}_W$ maps each pair of nodes to some integer (index of the weight) or zero (no connection). If a certain property is not fulfilled, e.g., if the distance between two nodes is too large or the type of the edge is invalid there should be no connection between the nodes. Thus, for $\mathcal{D} \subseteq 0 \cup \mathbb{N}$ being the set of all valid values for p values we require $\mathbf{R}_W(i,j) = 0$ if $p(i,j) \notin \mathcal{D}$. Moreover, we require $\mathbf{R}_W(i,j) = \mathbf{R}_W(k,l)$ if and only if $t(i,j) = t(k,l)$ and $p(i,j) = p(k,l) \in \mathcal{D}$. For the bias term we require that $\mathbf{R}_b$ maps each node to zero or some integer and two nodes i and j to the same integer if and only if $l(i) = l(j)$. Besides these requirements the exact value of the integer is not important as it is only the index of the weight in the set of learnable parameters. Of course in practice it is of advantage to use consecutive integers starting at 1 for the indices.

Proposition 1 *Let $f(-, \Theta, \mathbf{R})$ be a graph rule based layer and $x \in \mathbb{R}^{|V|}$ the input signal corresponding to a graph $G = (V, E)$ Then for every permutation π of the node order of G (permutation of the entries of x) it holds $f(\pi(x), \Theta, \mathbf{R}) = \pi(f(x, \Theta, \mathbf{R}))$, i.e., f is permutation equivariant.*

In particular, Proposition 1 shows that each l and p as defined above gives rise to a permutation equivariant rule based layer. Thus, finding a meaningful rule for graphs reduces to finding a meaningful labeling function l and property function p. In the following we focus on three different rule based layers that are based on well-known graph labeling functions.

Weisfeiler-Leman Layer Recent research has shown that Weisfeiler-Leman labeling is a powerful tool for graph classification [23,17,3,25]. Thus, we propose to use 1-Weisfeiler-Leman labels of iteration k as one option for l. The 1-Weisfeiler-Leman algorithm assigns in the k-th iteration to each node of a graph a label based on the structure of its local k-hop neighborhood, see [23] for the details[1]. For p we use the distance between two nodes, i.e., $p \equiv d$ and $\mathcal{D} \subset 0 \cup \mathbb{N}$ is the set of valid distances. We denote the induced rule by $\mathbf{R}_{WL_{k,\mathcal{D}}}$. For computational reasons in the experiments we restrict the maximum number of different Weisfeiler-Leman labels by L. We relabel the most frequent $L-1$ labels to $1, \ldots, L-1$ and set all others to L. The corresponding layer is denoted by $f_{WL_{k,\mathcal{D},L}}$.

Pattern Counting Layer Beyond labeling nodes via the Weisfeiler-Leman algorithm, it is a common approach to use subgraph isomorphism counting to distinguish graphs [4]. This is in fact necessary as the 1-Weisfeiler-Leman algorithm is not able to distinguish some types of graphs, for example circular skip link graphs [5] and strongly regular graphs [3,4]. Thus, we propose a node labeling function l based on pattern counting. The function p is the same as for the Weisfeiler-Leman layer. In general, subgraph isomorphism counting is a hard problem [6], but for the real-world and synthetic benchmark graph datasets

[1] Usually, Weisfeiler-Leman labels are represented via strings. For our purpose the strings are hashed to integers.

that are usually considered, subgraphs of size $k \in \{3, 4, 5, 6\}$ can be enumerated in a preprocessing step in a reasonable time, see Appendix C.2. Given a set of patterns, say $\mathcal{P}$, we compute all possible embeddings of these patterns in the graph dataset in a preprocessing step. Then for each pattern $P \in \mathcal{P}$ and each node $i \in V$ we count how often the node i is part of an embedding of P. Using those counts we define a labeling function $l : V \to \mathcal{L} \subseteq \mathbb{N}$ and two nodes $i, j \in V$ are mapped to the same label if and only if their counts are equal for all patterns in $\mathcal{P}$. Patterns that are often used in practice are small cycles, cliques, stars or paths. We denote the corresponding rule by $\mathbf{R}_{\mathcal{P}_{\mathcal{D}}}$. As for the Weisfeiler-Leman Rule we restrict the maximum number of different labels to some number L. The corresponding layer is denoted by $f_{\mathcal{P}_{\mathcal{D},L}}$.

The total number of learnable parameters for Weisfeiler-Leman or Pattern counting layer is bounded by $L \cdot L \cdot |\mathcal{D}|$ for the weight matrix and $|L|$ for the bias.

Aggregation Layer In contrast to the above layers we assume that $m = M$ and $n = |V|$. Let $l : V \to \mathcal{L}$ be an arbitrary labeling function, e.g., the atom labels in molecule graphs, the degree of the nodes or the Weisfeiler-Leman labels. We require the rule function $\mathbf{R}^M_{\text{Aggr}}$ associated with the weight matrix to assign each pair (n, i) with $i \in V$ and $n \in [M]$ an integer or zero based on n and $l(i)$. In fact, for each element of $\mathcal{L}$ the rule defines M different learnable parameters. The rule function $\mathbf{R}^M_{\text{Aggr}}$ associated with the bias is the identity, i.e., it represents an ordinary bias term with M learnable parameters. The corresponding layer is denoted by $f_{\mathbf{R}^M_{\text{Aggr}}}$. We use this layer as output layer because its output is a fixed dimensional vector of size $M \in \mathbb{N}$ independent of the input size. The total number of learnable parameters for the aggregation layer is bounded by $M \cdot |\mathcal{L}|$ for the weight matrix and M for the bias term.

Proposition 2 *The aggregation layer $f_{\mathbf{R}^M_{\text{Aggr}}}$ is permutation invariant, i.e., for any permutation π of the nodes of $G = (V, E)$ with corresponding input signal x it holds $f_{\mathbf{R}^M_{\text{Aggr}}}(\pi(x), \Theta, \mathbf{R}) = f_{\mathbf{R}^M_{\text{Aggr}}}(x, \Theta, \mathbf{R})$.*

3.4 Rule Graph Neural Networks (RuleGNNs)

The layers defined above are the building blocks of RuleGNNs. Each RuleGNN is a concatenation of different rule based layers from type *Weisfeiler-Leman* and *Pattern Counting* with different parameters followed by an *Aggregation Layer*. The input of the network is a signal $x \in \mathbb{R}^{|V|}$ corresponding to a graph $G = (V, E)$. We note that for simplicity we focus on one-dimensional signals but our approach also allows multidimensional signals, i.e., $x \in \mathbb{R}^{|V| \times d}$. The output of the network is a vector of fixed size $M \in \mathbb{N}$ determined by the aggregation rule where M is usually the number of classes of the graph classification task. The output can be also used as an intermediate vectorial representation of the graph or for regression tasks. Using Proposition 1 and Proposition 2 it follows directly that RuleGNNs are invariant under the chosen node order of the input signal. Note that RuleGNNs can be also used for node classification tasks by setting

$M = |V|$ or by omitting the aggregation layer. Moreover, we can show that the expressive power of RuleGNNs is at least as powerful as the underlying labeling function l and thus using an appropriate labeling function we can distinguish arbitrary non-isomorphic graphs.

Theorem 1 (Expressive Power of RuleGNNs) *For each non-isomorphic graphs G and G' it exists a RuleGNN $f(-, \Theta, \mathcal{R})$ that distinguishes G and G'.*

4 Experiments

We evaluate the performance of RuleGNNs on different real-world and synthetic benchmark graph dataset and compare the results to state-of-the-art algorithms. For comparability and reproducibility of the results, we make use of the experimental setup from [9]. For each graph dataset we perform a 10-fold cross validation, i.e., we use fixed splits of the dataset into 10 equally sized parts, and use 9 of them for training, parameter tuning and validation [2]. We then use the model that performs best on the validation set and report the performance on the previously unseen test set. We average three runs of the best model to decrease random effects. The standard deviation reported in the tables is computed over the results on the 10 folds.

Data and Competitors Selection A problem of several heavily used graph benchmark datasets like MUTAG or PTC [16] is that node and edge labels seems to be more important than the graph structure itself, i.e., there is no significant improvement over simple baselines [21]. Moreover, in case of MUTAG the performance of the model is highly dependent on the data split because of the small number of samples. Thus, in this work for benchmarking we choose DHFR, Mutagenicity, NCI1, NCI109, IMDB-BINARY and IMDB-MULTI from [16] because the structure of the graphs seems to play an important role, i.e., simple baselines [9,21] are significantly worse than more evolved algorithms. Additionally, we consider circular skip link graphs CSL [5] and some new synthetic benchmark graph datasets called LongRings, EvenOddRings and Snowflakes [18] to show that RuleGNNs can overcome limitations of ordinary graph neural networks. For more details on the datasets see Appendix C.1. For NCI1, IMDB-BINARY and IMDB-MULTI we use the same splits as in [9] and for CSL we use the splits as in [8] and a 5-fold cross validation. We evaluate the performance of the RuleGNNs on these datasets and compare the results to the baselines from [9] and [21] and the Weisfeiler-Leman subtree kernel (WL-Kernel) [23] which is one of the best performing graph classification algorithm besides graph neural networks. For comparison with state-of-the-art graph classification algorithms we follow [9] and compare to DGCNN [29], GIN [27] and GraphSAGE [11]. Additionally, we compare to the results of some recent state-of-the-art graph classification algorithms [2,3,4,25]. For the latter we use the results from the respective papers that are obtained using another evaluation setup.

[2] See https://github.com/fseiffarth/RuleGNNCode for the data splits and the code.

We introduce the following four new synthetic benchmark graph datasets to test the ability of graph neural networks to capture long range dependencies, to encode expert knowledge and to distinguish graphs that are not distinguishable by the 1-WL test.

LongRings The dataset consists of 1200 cycles of 100 nodes each and is designed to test the ability to detect long range dependencies. Four of the cycle nodes are labeled by $1, 2, 3, 4$ and all others by 0. The distance between each pair of the four nodes is exactly 25 or 50. The label of the graph is 0 if 1 and 2 have distance 50, 1 if 1 and 3 have distance 50 and 2 if 1 and 4 have distance 50. There are 400 graphs per class. The difficulty of the classification task is that information has to be propagated over a long distance. Regarding RuleGNNs this is very easy as we can define an appropriate rule.

EvenOddRings The dataset consists of 1200 cycles of 16 nodes each and is designed to test the ability to encode expert knowledge in the neural network architecture. The nodes in each graph are labeled from 0 to 15. The graph label is determined by labels of the nodes that have distance 8 respectively 4 to the node with label 0. We denote them by x resp. y, z. We have four cases: x is even and $y + z$ is even, x is even and $y + z$ is odd, x is odd and $y + z$ is even, x is odd and $y + z$ is odd. There are 300 graphs per class, i.e., each of the four cases. The expert knowledge we use is that the information has to be collected from nodes of distance 8 and 4 only.

EvenOddRingsCount The dataset consists of the same graphs as EvenOddRings but the graph labels are different. For all nodes and their opposite node (distance 8) in the circle the sum of the labels is computed. If there are more even sums than odd sums the graph is labeled by 0 and by 1 otherwise. There are 600 graphs per class. The expert knowledge we use is the information that only distance 8 is relevant.

Snowflakes The dataset consists of graphs proposed by [18] that are not distinguishable by the 1-WL test, see Figure 5 for an example. The dataset consists of circles of length 3 to 12 and at each circle node a graph from M_0, M_1, M_2 or M_3 is attached, see Figure 6 and [18] for the details. M_0, M_1, M_2 and M_3 are non-isomorphic graphs that are not distinguishable by the 1-WL test. One node in the circle is labeled by 1 and all other nodes are labeled by 0. The label of the graph is determined by the graph M_0, M_1, M_2 or M_3 that is attached to the circle node with label 1.

Experimental Settings and Resources All experiments were conducted on an AMD Ryzen 9 7950X 16-Core Processor with 128 GB of RAM. For the competitors we use the implementations from [9][3]. For the real-world datasets we tested different rules and combinations of the layers defined in Section 3.3. More details on the

[3] See https://github.com/fseiffarth/gnn-comparison for the code.

tested hyperparameters can be found in Appendix C.3. We always use tanh for activation and the Adam optimizer [13] with a learning rate of 0.05 (real-world datasets) resp. 0.1 (synthetic datasets). For the real-world datasets the learning rate was decreased by a factor of 0.5 after each 10 epochs. For the loss function we use the cross entropy loss. All models are trained for 50 (real-world) resp. 200 (synthetic) epochs and the batch size was set to 128. We stopped the training if the validation accuracy did not improve for 25 epochs.

4.1 Results

Real-World Datasets The results on the real-world datasets (Table 1) show that RuleGNNs are able to outperform the state-of-the-art graph classification algorithms in the setting of [9] even if we add all the additional label information that RuleGNNs use to the input features of the graph neural networks (see the (features) results in Table 1). This shows that the structural encoding of the additional label information is crucial for the performance of the graph neural networks and not replaceable by using additional input features. The Weisfeiler-Leman subtree kernel [23] is the best performing graph classification algorithm on NCI1, NCI109 and Mutagenicity but not on DHFR, IMDB-BINARY and IMDB-MULTI. For IMDB-BINARY and IMDB-MULTI our approach performs worse than the state-of-the-art graph classification algorithms that are not evaluated within the same experimental setup. This can be the result of the different evaluation setup or the fact that we do not used the best rule for these datasets.

Synthetic Datasets The results on the synthetic benchmark graph dataset (Table 2) show that the expressive power of RuleGNNs is higher than that of the standard message passing model. Moreover, the integration of expert knowledge in the form of rules leads to a significant improvement in the performance of the model. In fact, CLS and Snowflakes are not solvable by the message passing model because they are not distinguishable by the 1-WL test. The results on LongRings show that long range dependencies can be easily captured by RuleGNNs and also dependencies between nodes of different distances as in case of the EvenOddRings dataset can be easily encoded by appropriate rules.

Interpretability of RuleGNNs Each learnable parameter of RuleGNNs used for the weight matrices can be interpreted in terms of the importance of a connection between two nodes in a graph with respect to their labels and their shared property (in our case the distance). That is, each model provides the relevance of two nodes i, j in a graph with labels $l(i), l(j)$ and distance $d(i, j)$. In Figures 1 and 2 we see how the network has learned the importance of different connections between nodes for different distances and labels. The weights are visualized by arrows (thickness corresponds to the absolute value and the color to the sign). The biases are visualized by the nodes (size corresponds to the absolute value and the color to the sign). Figure 1 shows an example of the relevance of the weights for graphs from the DHFR and IMDB-BINARY datasets using the best model. We can see that in case of DHFR the RuleGNN has learned to pass

	NCI1	**NCI109**	**Mutagenicity**	**DHFR**	**IMDB-B**	**IMDB-M**
Baseline (NoG) [21]	69.2 ± 1.9	68.4 ± 2.2	74.8 ± 1.8	71.8 ± 5.3	71.9 ± 4.8	47.7 ± 4.0
WL-Kernel[23]	**85.2 ± 2.3**	**85.0 ± 1.7**	**83.8 ± 2.4**	83.5 ± 5.1	71.8 ± 4.5	51.9 ± 5.6
DGCNN[29]	76.4 ± 1.7	73.0 ± 2.4	77.0 ± 2.0	72.6 ± 3.1	69.2 ± 3.0	45.6 ± 3.4
DGCNN (features)	73.6 ± 1.0	72.5 ± 1.5	76.3 ± 1.2	76.1 ± 3.4	69.1 ± 3.5	45.8 ± 2.9
GraphSage[11]	76.0 ± 1.8	77.1 ± 1.8	79.8 ± 1.1	80.7 ± 4.5	68.8 ± 4.5	47.6 ± 3.5
GraphSage (features)	79.4 ± 2.2	78.6 ± 1.6	80.1 ± 1.3	82.4 ± 3.9	69.7 ± 3.1	46.6 ± 4.8
GIN[27]	80.0 ± 1.4	79.7 ± 2.0	81.9 ± 1.4	79.1 ± 4.4	71.2 ± 3.9	48.5 ± 3.3
GIN (features)	77.3 ± 1.8	77.7 ± 2.0	80.6 ± 1.3	81.8 ± 5.1	70.9 ± 3.8	48.3 ± 2.7
GSN (paper) [4]	83.5 ± 2.3	-	-	-	77.8 ± 3.3	54.3 ± 3.3
CIN (paper) [2]	83.6 ± 1.4	84.0 ± 1.6	-	-	75.6 ± 3.7	52.7 ± 3.1
SIN (paper) [3]	82.7 ± 2.1	-	-	-	75.6 ± 3.2	52.4 ± 2.9
PIN (paper) [25]	85.1 ± 1.5	84.0 ± 1.5	-	-	76.6 ± 2.9	-
RuleGNN	82.8 ± 2.0	83.2 ± 2.1	81.5 ± 1.3	**84.3 ± 3.2**	**75.4 ± 3.3**	**52.0 ± 4.3**

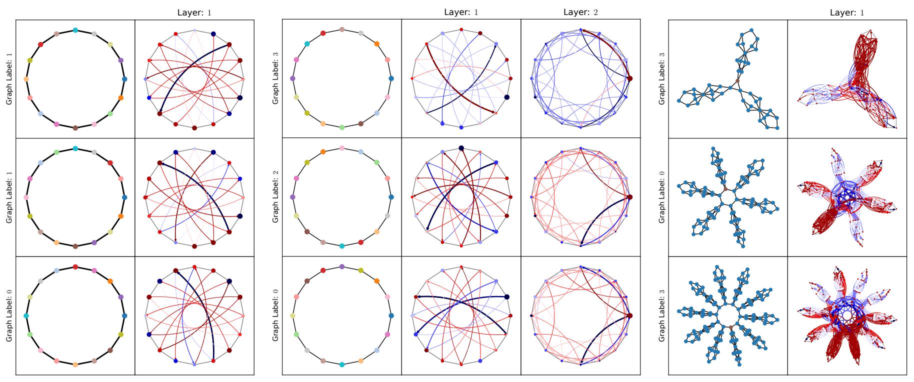
Table 1: Test set performance of several state-of-the-art graph classification algorithms averaged over three different runs and 10 folds. The ± values report the standard deviation over the 10 folds. The overall best results are colored red and the best ones obtained for the fair comparison from [9] are in bold. The (features) variants of the algorithms use the same information as the RuleGNN as input features additionally to node labels. The (paper) results are taken from the respective papers using another experimental setup.

	LongRings	EvenOddRings	EvenOddRingsCount	CSL	Snowflakes
Baseline (NoG) [21]	30.17 ± 3.2	22.25 ± 3.0	47.9 ± 3.9	10.0 ± 0.0	27.3 ± 5.3
WL-Kernel [23]	**100.0 ± 0.0**	26.83 ± 4.2	47.8 ± 4.3	10.0 ± 0.0	27.9 ± 4.1
DGCNN [29]	29.9 ± 2.6	28.4 ± 2.5	59.1 ± 5.2	10.0 ± 0.0	26.0 ± 3.3
GraphSAGE [11]	29.8 ± 2.8	24.9 ± 2.7	51.3 ± 1.9	10.0 ± 0.0	25.0 ± 1.8
GIN [27]	32.0 ± 3.1	26.8 ± 2.5	51.0 ± 3.7	10.0 ± 0.0	24.5 ± 2.2
RuleGNN	99.0 ± 3.3	**90.2 ± 7.2**	**100.0 ± 0.0**	**100.0 ± 0.0**	**97.9 ± 3.2**

Table 2: Test set performance of several state-of-the-art graph classification algorithms averaged over three different runs and 10 folds. The ± values report the standard deviation over the 10 folds. The best results are highlighted in bold.

messages from the outer nodes to ring nodes. Some ring nodes seem to be more important than others. It is an interesting open question if these connections can be interpreted in a chemical context. In case of the IMDB-BINARY dataset we can see that the RuleGNN has learned to pass messages to some specific nodes. It would be interesting to further investigate if these nodes have a specific meaning in the context of the dataset. Figure 2 shows an example of the learned parameters for our synthetic datasets. Considering the dataset RingEvenOdd in Figure 2b we see that in the first layer the RuleGNN passes the messages between opposite nodes as given by the rule. In the second layer it has learned the relevant information, i.e., to collect the information from the nodes that have distance 4 to the node with label 0 (dark blue node). All other connections of distance 4 have a smaller weight, i.e., are less important. For the Snowflakes dataset Figure 2c we see that the RuleGNN has learned to distinguish between the four different subgraphs M_0, M_1, M_2 and M_3 glued to the central circle by looking at the learned parameters. Indeed, each subgraph from type M_0, M_1, M_2 and M_3 can be identified by the characteristics given by the learned parameters.

This shows that the RuleGNN we have used for the Snowflakes dataset is more powerful than the 1-WL test.



(a) EvenOddCount (b) EvenOddRings (c) Snowflakes

Fig. 2: Visualization of the learned parameters for the EvenOddRingsCount (a), EvenOddRings (b) and Snowflakes (c) dataset. The first column shows the graphs and the colors of the nodes represent the different node labels. The other columns show the learned weights and biases for the respective rule based layer. The message passing weights are visualized by arrows (thicker for higher absolute values) and the biases are visualized by the size of the node (red for positive and blue for negative weights).

5 Concluding Remarks and Outlook

We have introduced rule based layers that dynamically arrange the learnable parameters in the weight matrices and bias vectors according to a formal rule. Using rule based layers for graph classification we are able to overcome some of the limitations of classical graph neural networks such as long range dependencies, varying graph sizes, restricted expressive power and lack of interpretability. The flexibility of our approach using arbitrary rules to adapt the architecture to the specific problem at hand leads to many interesting research questions. As each learnable parameter can be related to specific nodes in a graph, the question is if learned knowledge from one graph dataset can be transferred to another dataset. In the experiments we tested only a few possible rules, but in fact there are many more rules that can be used to improve the performance of RuleGNNs. One question is how to find the best rule for a given graph dataset and if it is possible to directly learn the rules from the data. Our proposed method is not limited to graph classification but can be used for other tasks as well. Thus, it would be interesting to see how rule based layers perform on other tasks such as node classification or different data structures such as text or images.

References

1. Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 21–29. PMLR, 2019.
2. Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Liò, Guido F. Montúfar, and Michael M. Bronstein. Weisfeiler and lehman go cellular: CW networks. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 2625–2640, 2021.
3. Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F. Montúfar, Pietro Lió, and Michael M. Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1026–1037. PMLR, 2021.
4. Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(1):657–668, 2023.
5. Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Comb.*, 12(4):389–410, 1992.
6. Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.
7. Zdenek Dvorák. On recognizing graphs by numbers of homomorphisms. *J. Graph Theory*, 64(4):330–342, 2010.
8. Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *J. Mach. Learn. Res.*, 24:43:1–43:48, 2023.
9. Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *ArXiv*, abs/1912.09893, 2019.
10. Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.
11. William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Neural Information Processing Systems*, 2017.
12. Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(11):7436–7456, 2022.
13. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning*

Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

14. Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
15. Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3538–3545. AAAI Press, 2018.
16. Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.
17. Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019.
18. Harish G. Naik, Jan Polster, Raj Shekhar, Tamás Horváth, and György Turán. Iterative graph neural network enhancement via frequent subgraph mining of explanations, 2024.
19. Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. k-hop graph neural networks. *Neural Networks*, 130:195–205, 2020.
20. Pablo Sánchez-Martín, Kinaan Aamir Khan, and Isabel Valera. Improving the interpretability of GNN predictions through conformal-based graph sparsification. *CoRR*, abs/2404.12356, 2024.
21. Till Hendrik Schulz and Pascal Welke. On the necessity of graph kernel baselines. 2019.
22. Florian Seiffarth. Rule based learning with dynamic (graph) neural networks. https://www.mlai.cs.uni-bonn.de/en/paper/seiffarth/rulenn.pdf 2024.
23. Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.
24. Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 29–38. IEEE Computer Society, 2017.
25. Quang Truong and Peter Chin. Weisfeiler and lehman go paths: Learning topological features via path complexes. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 15382–15391. AAAI Press, 2024.

26. Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2017.
27. Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.
28. Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9240–9251, 2019.
29. Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4438–4445. AAAI Press, 2018.
30. Zhenpeng Zhou and Xiaocheng Li. Graph convolution: A high-order and adaptive approach. *arXiv: Learning*, 2017.

A Proofs

Proof (Proposition 1). Using the definition of rule based layers (1) we have that

$$\pi(f(x, \Theta, \mathbf{R})) = \pi(\sigma(W_{\mathbf{R}_W(x)} \cdot x + b_{\mathbf{R}_b(x)})) \ .$$

The entries of the weight matrix and the bias term are given by the rule functions $\mathbf{R}_W$ and $\mathbf{R}_b$, see (2), which depend on the input signal x, i.e., the node order of the graph. Thus, by definition the permutation of the input signal $\pi(x)$ permutes the entries of the corresponding weight matrix and the bias term in the same way compared to the original input signal x. Therefore, the result of the multiplication of the permuted weight matrix with the permuted input signal is the same as the permutation of the result of the multiplication of the original weight matrix with the original input signal and it follows

$$\pi(\sigma(W_{\mathbf{R}_W(x)} \cdot x + b_{\mathbf{R}_b(x)})) = \sigma(W_{\mathbf{R}_W(\pi(x))} \cdot \pi(x) + b_{\mathbf{R}_b(\pi(x))}) = f(\pi(x), \Theta, \mathbf{R})$$

which completes the proof.

Proof (Proposition 2). Using the definitions of the aggregation rule, (3) and (4) it follows that node permutations permute the rows of the weight matrix and thus have no effect on the bias vector. In fact, permutations of the rows of the weight matrix do not change the result of the multiplication of the weight matrix with the input signal. Thus, the result of the aggregation layer is invariant under permutations of the nodes of the graph.

Proof (Theorem 1). The expressive power of the RuleGNNs is based on the expressive power of the underlying labeling function l. Indeed, we will show that a RuleGNN is at least as powerful as the labeling function l. Let l be a labeling function that can distinguish G and G' by counting the occurrences of the labels, e.g., the $(k+1)$-WL labels where k is the maximum of the treewidths of G and G' [7]. Now, consider the RuleGNN that consists only of the aggregation layer $f_{\mathbf{R}^M_{\text{Aggr}}}$ with $M = 1$ based on the labeling function l. Without loss of generality we assume that each entry of the input signals x resp. y corresponding to G resp. G' is equal to 1. Then $f_{\mathbf{R}^M_{\text{Aggr}}}(x)$ resp. $f_{\mathbf{R}^M_{\text{Aggr}}}(y)$ is equal to the sum of the learnable parameters corresponding to the labels of the nodes of G resp. G'. By assumption l can distinguish G and G' by counting the occurrences of the labels and hence also the above defined RuleGNN can distinguish G and G'.

B Example: RuleGNNs for Molecule Graphs

Assume the task is to learn a property of a molecule based on its graph structure. In this example we present a RuleGNN that is a concatenation of two very simple rule based layers. The advantage of rule based layers and hence also RuleGNNs is that they encode the graph structure (in this example the structure of two molecules) directly into the neural network. Moreover, the input data

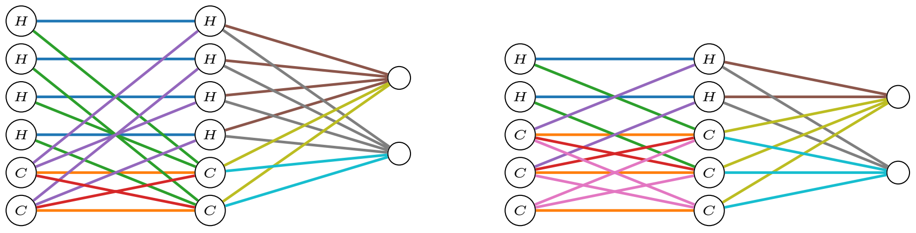


Fig. 3: Information propagation in a simple two layer RuleGNN based on the molecule graphs of ethylene (left) and cyclopropenylidene (right) and the rules $\mathbf{R}_{\text{Mol}}$ (5) and $\mathbf{R}^k_{\text{Aggr}}$ (6). The input signal is propagated from left to right. The graph nodes represent the neurons of the neural network. Edges of the same color denote shared weights in a layer.

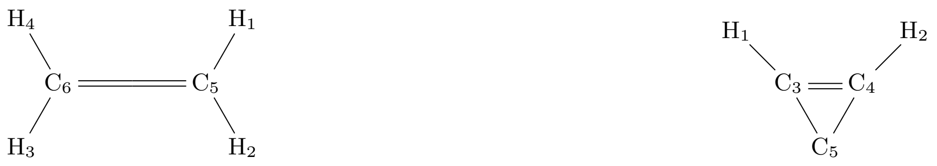


Fig. 4: Molecule graphs of ethylene (left) and cyclopropenylidene (right). The indices denote the order of the nodes.

can be arbitrary molecule graphs and the output is a vector of fixed size $k = 2$ that encodes the property of the molecule or some intermediate vectorial representation. In this example we consider the molecule graphs of ethylene and cyclopropenylidene given in Figure 4 together with their corresponding input signals $x \in \mathbb{R}^6$ and $y \in \mathbb{R}^5$. The atoms of the molecules (hydrogen H and carbon C) correspond to the nodes of a graph and the bonds to the edges. The atom labels and the bond types (*single* and *double*) can be seen as additional information $\mathbf{I}$ that is known about the input samples. The graph nodes are indexed via integers in some arbitrary but fixed order and the atoms corresponding to the graph nodes are given by the labeling function $l : V \to \{H, C\}$.

The RuleGNN consists of two rule based layers named $f_1(-, \Theta_1, \mathbf{R}_{\text{Mol}})$ and $f_2(-, \Theta_2, \mathbf{R}^2_{\text{Aggr}})$ with learnable parameters $\Theta_1 = \{w_1, \ldots, w_6\}$ and $\Theta_2 = \{w'_1, \ldots, w'_4\}$ and the following rule functions $\mathbf{R}_{\text{Mol}}$ and $\mathbf{R}^2_{\text{Aggr}}$. For some graph

$G = (V, E)$ and its corresponding input signal z we define $\mathbf{R}_{\text{Mol}}$ as follows:

$$\mathbf{R}_{\text{Mol}}(z) : [|V|] \times [|V|] \longrightarrow \{0\} \cup [6]$$

$$(i, j) \qquad \mapsto \begin{cases} 1 & \text{if } i = j \text{ and } l(i) = H \\ 2 & \text{if } i = j \text{ and } l(i) = C \\ 3 & \text{if } (i, j) \text{ is an edge (-), } l(i) = H, l(j) = C \\ 4 & \text{if } (i, j) \text{ is an edge (-),} l(i) = C, l(j) = H \\ 5 & \text{if } (i, j) \text{ is an edge (-),} l(i) = l(j) = C \\ 6 & \text{if } (i, j) \text{ is an edge (=),} l(i) = l(j) = C \\ 0 & \text{o.w.} \end{cases} \tag{5}$$

For some graph $G = (V, E)$ and its corresponding input signal z we define $\mathbf{R}_{\text{Aggr}}$ as follows:

$$\mathbf{R}^2_{\text{Aggr}}(z) : [2] \times [|V|] \longrightarrow \{0\} \cup [4]$$

$$(i, j) \qquad \mapsto \begin{cases} i & l(j) = H \\ i + 2 & l(j) = C \\ 0 & \text{o.w.} \end{cases} \tag{6}$$

Note that $\mathbf{R}_{\text{Mol}}$ and $\mathbf{R}^2_{\text{Aggr}}$ are not restricted to the two molecules from above but can be applied to arbitrary molecule graphs. Indeed, applying it to molecules with atom labels different from H or C makes the rules less powerful, i.e., it should be adapted to the type of molecules. Using the definition (3) of weight distribution defined by the rule function we can construct the weight matrices $W_{\mathbf{R}_{\text{Mol}}(x)}, W_{\mathbf{R}^2_{\text{Aggr}}(x)}$ for the ethylene graph and $W_{\mathbf{R}_{\text{Mol}}(y)}, W_{\mathbf{R}_{\text{Aggr}}(y)}$ for the cyclopropenylidene graph as follows:

$$W_{\mathbf{R}_{\text{Mol}}(x)} = \begin{pmatrix} w_1 & 0 & 0 & 0 & w_3 & 0 \\ 0 & w_1 & 0 & 0 & w_3 & 0 \\ 0 & 0 & w_1 & 0 & 0 & w_3 \\ 0 & 0 & 0 & w_1 & 0 & w_3 \\ w_4 & w_4 & 0 & 0 & w_2 & w_5 \\ 0 & 0 & w_4 & w_4 & w_5 & w_2 \end{pmatrix} \; W_{\mathbf{R}^2_{\text{Aggr}}(x)} = \begin{pmatrix} w'_1 & w'_1 & w'_1 & w'_1 & w'_3 & w'_3 \\ w'_2 & w'_2 & w'_2 & w'_2 & w'_4 & w'_4 \end{pmatrix}$$

$$W_{\mathbf{R}_{\text{Mol}}(y)} = \begin{pmatrix} w_1 & 0 & w_3 & 0 & 0 \\ 0 & w_1 & 0 & w_3 & 0 \\ w_4 & 0 & w_2 & w_6 & w_5 \\ 0 & w_3 & w_6 & w_2 & w_5 \\ 0 & 0 & w_5 & w_5 & w_2 \end{pmatrix} \qquad W_{\mathbf{R}^2_{\text{Aggr}}(y)} = \begin{pmatrix} w'_1 & w'_1 & w'_3 & w'_3 & w'_3 \\ w'_2 & w'_2 & w'_4 & w'_4 & w'_4 \end{pmatrix}$$

Combining the two rule based layers we obtain the RuleGNN and the forward propagation is given by $\sigma(W_{\mathbf{R}_{\text{Aggr}}(x)} \cdot \sigma(W_{\mathbf{R}_{\text{Mol}}(x)} \cdot x))$ for the ethylene graph and $\sigma(W_{\mathbf{R}_{\text{Aggr}}(y)} \cdot \sigma(W_{\mathbf{R}_{\text{Mol}}(y)} \cdot y))$ for the cyclopropenylidene graph.

Note that the forward propagation of the layer corresponding to the rule $\mathbf{R}_{\text{Mol}}$ is kind of a multiplication with a weighted adjacency matrix of the graph where the weights of the adjacency matrix are given by the learnable parameters,

Dataset	#Graphs	#Nodes			#Edges			Diameter			#Node Labels	#Classes
		max	avg	min	max	avg	min	max	avg	min		
NCI1	4 110	111	29.9	3	119	32.3	2	45	11.5	0	37	2
NCI109	4 127	111	29.7	4	119	32.1	3	61	11.3	0	38	2
Mutagenicity	4 337	417	30.3	4	112	30.8	3	41	6.3	0	14	2
DHFR	756	71	42.4	20	73	44.5	21	22	14.6	8	9	2
IMDB-BINARY	1 000	136	19.8	12	1249	96.5	26	2	1.9	1	1	2
IMDB-MULTI	1 500	89	13.0	7	1467	65.9	12	2	1.5	1	1	3

Table 3: Details of the real-world datasets [16] used in the experiments.

Dataset	#Graphs	#Nodes			#Edges			Diameter			#Node Labels	#Classes
		max	avg	min	max	avg	min	max	avg	min		
LongRings	1 200	100	100.0	100	100	100.0	100	50	50.0	50	5	3
EvenOddRings	1 200	16	16.0	16	16	16.0	16	8	8.0	8	16	4
EvenOddRingsCount	1 200	16	16.0	16	16	16.0	16	8	8.0	8	16	2
CSL [5]	150	41	41.0	41	82	82.0	82	10	6.0	4	1	10
Snowflakes	1 000	180	112.5	45	300	187.5	75	18	15.5	13	2	4

Table 4: Details of the synthetic datasets used in the experiments.

see also Figure 3. In contrast to adjacency matrices the weight matrix is not necessary symmetric. The computation graph induced by the weight matrices exactly represent the graph structure while the edge weights are shared across the network using the rule, see Figure 3. Note that also edge labels (e.g., atomic bonds) can be taken into account by increasing the size of the weight set. Moreover, it is possible to include bigger neighborhoods, i.e., all nodes reachable by k-hops. Of course using other information of the graph (e.g., substructures (such as circles or cliques), node degrees, connections not depicted by edges) more complicated rules such as the Weisfeiler-Leman rule and Pattern Counting rules can be used.

C Evaluation Details

In this section we provide some additional details on the benchmark datasets and the evaluation of the RuleGNNs.

C.1 Dataset Details

In this section we provide additional details on the datasets used in the experiments. Tables 3 and 4 provide an overview of the real-world and synthetic datasets including the number of graphs, the number of nodes, the number of edges, the diameter, the number of node labels and the number of classes. Figure 5 shows an example of the Snowflakes dataset and Figure 6 shows the graphs M_0, M_1, M_2 and M_3 that are part of the snowflakes dataset and not distinguishable by the 1-WL test.

C.2 Preprocessing and Training Details

Table 5 shows more details of training of RuleGNNs on the different datasets. In particular, we see that except for the DHFR dataset we need less than 12

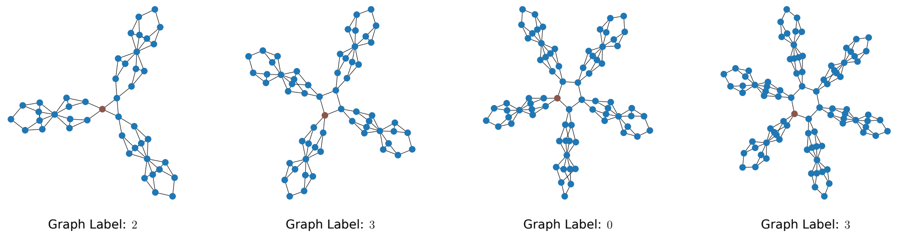


Fig. 5: Example graphs from the *Snowflakes* dataset. The brown node in the circle is labeled by 1 and the other nodes by 0. The label of the graph is determined by the subgraph attached to the brown node.

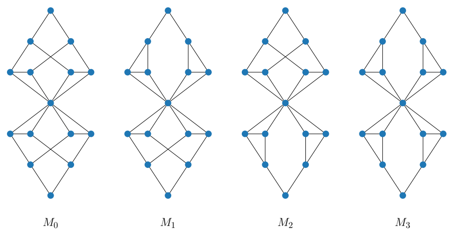


Fig. 6: The graphs M_0, M_1, M_2 and M_3 [18] that are not distinguishable by the 1-WL test.

epochs on average to reach the best result. This shows that our approach is very efficient and converges quickly. At the first glance the average time per epoch seems to be very high which has two reasons. One is also mentioned in [12] that there is a gap between the theoretical and practical runtime of dynamic neural networks because the implementation in PyTorch is not optimized for dynamic neural networks. The other reason is that our computations run in parallel, i.e., we are able to run all the three runs and 10 folds in parallel on the same machine which produces some overhead but is more efficient than running the experiments sequentially. As stated above the preprocessing times (Table 5) are not relevant for the experiments as they are only needed once. The third column shows the time needed to compute all the pairwise distances between the nodes of the graph. The fourth column shows the time needed to compute the node labels used for the best model. The most preprocessing time is needed for IMDB-BINARY and IMDB-MULTI because the graphs are much denser than the other datasets. For the synthetic datasets except for CSL and Snowflakes we do not need any label preprocessing time as the original node labels are used.

C.3 Architecture Details

Table 6 provides an overview of the different architectures used in the experiments that achieved the best results on the validation set. One advantage of our

Dataset	Best Epoch	Avg. Epoch (s)	Preproc. Distances (s)	Preproc. Labels (s)	#Graphs
NCI1	8.3 ± 5.3	377.1 ± 20.7	2.0	11.9	4 110
NCI109	6.4 ± 2.9	386.7 ± 1.9	2.4	13.2	4 127
Mutagenicity	10.1 ± 4.1	575.8 ± 66.4	2.2	15.2	4 337
DHFR	24.1 ± 14.6	44.4 ± 9.0	0.7	3.1	756
IMDB-BINARY	12.3 ± 4.6	24.3 ± 0.9	0.2	206.5	1 000
IMDB-MULTI	7.7 ± 3.5	19.6 ± 1.3	0.2	195.0	1 500
LongRings	195.2 ± 15.1	0.7 ± 0.2	6.6	-	1 200
EvenOddRings	177.1 ± 15.2	1.2 ± 0.3	0.2	-	1 200
EvenOddRingsCount	200.0 ± 0.0	0.5 ± 0.1	0.1	-	1 200
CSL	50.0 ± 0.0	1.6 ± 0.0	0.1	11.8	150
Snowflakes	192.7 ± 18.9	0.5 ± 0.1	7.1	116.8	1 000

Table 5: Runtimes and preprocessing times of the different datasets used in the experiments. All values are averaged over the best runs. The first column shows the best epoch (highest validation accuracy), the second the average time per epoch, the third the time needed to compute all the pairwise distances between the nodes of the graph, the fourth the time needed to compute the node labels used for the best model and the last the number of graphs in the dataset.

approach is that messages can be passed over long distances. Hence, except for the EvenOddRings dataset we used only one layer and the output layer. In case of NCI1, NCI109, Mutagenicity it turns out that the best model uses the Weisfeiler-Leman rule with $k = 2$ iterations. We restricted the number of maximum labels considered to 500 which results in 250000 learnable parameters for the weight matrix and 500 for the bias vector. For the output layer we used the bound of 50000 learnable parameters which was larger than the number of different Weisfeiler-Leman labels in the second iteration. Interestingly, for NCI1 and NCI109 the best validation accuracy was achieved if considering node pairs with distances from 1 to 10, while in case of Mutagenicity the best model uses node pairs with distances from 1 to 3. We also tested different small patterns, e.g., simple cycles, but they did not improve the results. For DHFR the best model uses simple cycles with length at most 10 as patterns for the output layer. We also tested the Weisfeiler-Leman rule in this case but the validation accuracy was lower. For IMDB-BINARY and IMDB-MULTI the best model uses the patterns simple cycles with length at most 10, the triangle and a single edge. Note that counting the embedding of a single edge as pattern is equivalent to the degree of the node. We also tested the Weisfeiler-Leman rule but the validation accuracy was lower[4]. As a next step it would be interesting to consider more rules, rules that come from expert knowledge or also deeper architectures with more rule based layers concatenated. Regarding the number of learnable parameters we would like to mention that the number is relatively high but lots of parameters are not used in the weight matrix. Hence, it might be possible to prune the set of learnable parameters by removing those that are not used or those that have a small absolute value.

For the synthetic datasets we use "expert knowledge" to define the rules. Hence we did not tested other rules than those in Table 6. For LongRings,

[4] See https://github.com/fseiffarth/RuleGNNCode for a full list of tested hyperparameters.

EvenOddRings and EvenOddRingsCount we used the original node labels for the rule based layers. In case of EvenOddRings we used two layers. The first layer considers only node pairs with distance 8 and collects all the necessary information of opposite nodes. The second layer that considers only node pairs with distance 4 and collects the information of the nodes that are 4 hops away from the nodes with label 0, see also Figure 2. For CSL we used as patterns all simple cycles with length at most 10. For the Snowflakes dataset we used the patterns, cycle of length 4 and 5 and collect the information of all nodes that have pairwise distance 3. In this way the RuleGNN is able to distinguish the graphs M_0, M_1, M_2 and M_3 that are not distinguishable by the 1-WL test. For the output layer we used the Weisfeiler-Leman rule with $k = 2$ iterations to collect the relevant information.

Dataset	Rules	Hyperparameters			#Learnable Parameters
		k	$\mathcal{D}$	L	per Layer
NCI1	wl	2	{1,. . . ,10}	500	2 500 500
	wl	2	-	50000	4 220
NCI109	wl	2	{1,. . . ,10}	500	2 500 500
	wl	2	-	50000	4 336
Mutagenicity	wl	2	{1,. . . ,3}	500	750 500
	wl	2	-	50000	4 972
DHFR	wl	2	{1,. . . ,6}	500	1 382 880
	pattern: (simple_cycles≤ 10)	-	-	-	112
IMDB-BINARY	pattern: (triangle, edge)	-	{1,2}	-	963 966
	pattern: (induced_cycles≤ 5)	-	-	-	990
IMDB-MULTI	pattern: (triangle, edge)	-	{1,2}	-	551 775
	pattern: (triangle, edge)	10	-	-	1 578
LongRings	labels	-	{25}	-	30
	labels	-	-	-	18
EvenOddRings	labels	-	{8}	-	272
	labels	-	{4}	-	272
	labels	-	-	-	68
EvenOddRingsCount	labels	-	{8}	-	272
	labels	-	-	-	34
CSL	pattern: (simple_cycles≤ 10)	-	{1}	-	8930
	pattern: (simple_cycles≤ 10)	-	-	-	950
Snowflakes	pattern: (cycle_4, cycle_5)	-	{3}	-	90
	wl	2	-	-	20

Table 6: Best architectures per dataset. The column *Rule* shows the type of rule used in the model, *wl* stands for the Weisfeiler-Leman labeling, *pattern* for the pattern based labeling and *labels* for the original node labels. The last layer is always an aggregation layer. While the others are Weisfeiler-Leman layers or Pattern Counting layers based on the labeling of the nodes. The column *Hyperparameters* shows the hyperparameters used in the model, k is the number of iterations of the Weisfeiler-Leman rule, $\mathcal{D}$ is the set of valid pairwise distances considered and L is the bound for the number of different node labels considered. The column *#Learnable Parameters* shows the number of learnable parameters in the model.