Machine Learning (2021) 110:1609–1636
https://doi.org/10.1007/s10994-021-05966-z

Incorporating symbolic domain knowledge into graph neural networks

Tirtharaj Dash[1] · **Ashwin Srinivasan**[1] · **Lovekesh Vig**[2]

Received: 15 February 2020 / Revised: 11 January 2021 / Accepted: 19 February 2021 /
Published online: 13 June 2021

Abstract

Our interest is in scientific problems with the following characteristics: (1) Data are naturally represented as graphs; (2) The amount of data available is typically small; and (3) There is significant domain-knowledge, usually expressed in some symbolic form (rules, taxonomies, constraints and the like). These kinds of problems have been addressed effectively in the past by symbolic machine learning methods like Inductive Logic Programming (ILP), by virtue of 2 important characteristics: (a) The use of a representation language that easily captures the relation encoded in graph-structured data, and (b) The inclusion of prior information encoded as domain-specific relations, that can alleviate problems of data scarcity, and construct new relations. Recent advances have seen the emergence of deep neural networks specifically developed for graph-structured data (Graph-based Neural Networks, or GNNs). While GNNs have been shown to be able to handle graph-structured data, less has been done to investigate the inclusion of domain-knowledge. Here we investigate this aspect of GNNs empirically by employing an operation we term *vertex-enrichment* and denote the corresponding GNNs as *VEGNN*s. Using over 70 real-world datasets and substantial amounts of symbolic domain-knowledge, we examine the result of vertex-enrichment across 5 different variants of GNNs. Our results provide support for the following: (a) Inclusion of domain-knowledge by vertex-enrichment can significantly improve the performance of a GNN. That is, the performance of *VEGNN*s is significantly better than *GNN*s across all GNN variants; (b) The inclusion of domain-specific relations constructed using ILP improves the performance of *VEGNNs*, across all GNN variants. Taken together, the results provide evidence that it is possible to incorporate symbolic domain knowledge into a GNN, and that ILP can play an important role in providing high-level relationships that are not easily discovered by a GNN.

Editors: Nikos Katzouris, Alexander Artikis, Luc De Raedt, Artur d'Avila Garcez, Ute Schmid, Sebastijan Dumančić, Jay Pujara.

✉ Tirtharaj Dash
tirtharaj@goa.bits-pilani.ac.in

Ashwin Srinivasan
ashwin@goa.bits-pilani.ac.in

[1] Department of Computer Science and Information Systems, APP Centre for Artificial Intelligence Research, BITS Pilani, K.K. Birla Goa Campus, Goa, India

[2] TCS Innovation Labs, New Delhi, India

1 Introduction

Industrialising scientific discovery, in the manner demonstrated by the Robot Scientist Project (King et al. 2004) uses machine learning programs as scientific assistants. At the very least, this would appear to require machine learning methods that are able to (a) cope with data that have some inherent structure, in the form of entities and relations; and (b) construct good predictive models by effectively drawing on any existing scientific knowledge thought to be relevant. A really useful assistant would have to do more. A wish-list would include identifying the best explanation for a prediction based on what is known; suggesting hidden variables or mechanisms which could improve the prediction; and proposing experiments to test the hypotheses. The Robot Scientist Project showed ways to achieve each of these in some measure with Inductive Logic Programming (ILP). Recent rapid gains in neural-network technology suggest that deep networks could form the basis of extremely powerful predictive models, which is clearly relevant to the construction of an effective scientific assistant. Here we investigate the performance of state-of-the-art deep networks specifically designed to analyse graph-structured data. A substantial number of applications addressed by ILP belong to this category of data (see, for example, King et al. 1996; Srinivasan and King 1999; Faruquie et al. 2012). There are at least three good reasons to investigate if graph neural networks, or GNNs, are able to incorporate domain-knowledge. First, studies with ILP have repeatedly shown that inclusion of domain-knowledge can make substantial difference to predictive performance. Furthermore, a recent report on Artificial Intelligence (AI) for Science identifies incorporating domain-knowledge in AI as one of the three Grand Challenges facing the application of (AI Stevens et al. 2020). Deep learning methods based on neural networks have not focused on this, relying instead on their internal computational machinery to construct higher-level concepts automatically from the raw data. The ILP experience suggests otherwise, and we would like to know if this applies to GNNs. Second, symbolic encodings of domain knowledge are both natural and flexible ways of encoding prior knowledge. ILP systems implemented as logic programs have been the pre-eminent form of machine learning for using such knowledge. Despite extremely efficient implementations of logic programming, the significant world-wide effort into the development of deep learning tools that have resulted in highly efficient implementations that exploit the processing capabilities of graphics processing units (GPUs). A GNN capable of including symbolic domain knowledge could provide an efficient way of constructing predictive models. Thirdly, to the best of our knowledge, GNN applications to date have been restricted to simple node-and-edge features, and have not attempted to encode any significant domain-knowledge. The real-world problems we examine in this paper have very extensive amounts of domain information, resulting from many years of academic and industrial effort into the use of ILP.

In this paper, we restrict the investigation to the problem of prediction, which we see as a necessary first step in the development of automated scientific assistants. Facilities for explanations and experiment-proposal using GNN models are conceptually harder, are deferred to future work. We assess the use of domain-knowledge using a sample of over 70 datasets, containing over 200,000 data instances. The datasets refer to problems in a broad category known as structure-activity prediction. Each data instance is, therefore, a molecule, which is naturally represented as a graph.[1] For this class of problems,

[1] In fact, GNNs were originally tested with molecular datasets (Baskin et al. 1997).

we now have a sufficiently large body of domain-knowledge encoded in human-understandable symbolic relations. This allows us to perform a case-study on the inclusion of symbolic relations by GNNs. The principal contributions of the paper are as follows:

- To the field of graph neural networks, the paper presents a large-scale empirical study using real-world datasets on the inclusion of domain-knowledge. To the best of our knowledge, the number of graphs used and the number of relations encoding domain-knowledge are the most extensive to date.
- To the field of inductive logic programming, the paper demonstrates a continuing case for the usefulness of ILP on relational learning tasks, despite the development of very efficient deep neural networks specifically designed for a specific form of relational data.
- To the field of neuro-symbolic modelling, the technique of vertex-enrichment described in the paper provides a simple but effective way of incorporating symbolic relations into graph-based neural networks.

The rest of the paper is organised as follows. In Section 2, we describe vertex-enrichment in graphs and a set of practical considerations arising from the developed algorithms. In Section 3, we describe our aims, data and background knowledge, the specifics of the methodology, and the obtained results. Section 4 lists related works, and Section 5 concludes the paper.

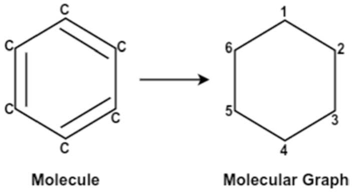
2 Graph neural networks (GNNs)

GNNs are primarily developed for learning from data represented as graphs. For completeness, we include some basic definitions first.

Definition 1 **(Graphs)** A graph G is a pair (V, E) where V is a set of vertices, E is a set of edges and a subset of $V \times V$. A graph is said to be undirected if for every $(v_i, v_j) \in E$, (v_j, v_i) in E.

We will be concerned in this paper with undirected graphs. We note that for such graphs, E can be represented more compactly as a set consisting of 1- or 2-element subsets of V. We will return to this later, as we extend the consideration to hypergraphs. For molecular graphs, of the kind considered here, self-loops do not occur.

Example 1 **Molecules as graphs.** A benzene ring (shown below) can be represented as a graph, in which vertices correspond to atoms and edges correspond to bonds (McNaught et al. 1997).



The graph-representation of the molecule on the left is:

$$(\{1,2,3,4,5,6\},\{(1,2),(2,1),(2,3),(3,2),(3,4),(4,3),(4,5),(5,4),(5,6),(6,5),(6,1),(1,6)\})$$

We will need the concept of the *neighbourhood* of a vertex in an undirected graph:[2]

Definition 2 **(Neighbourhood)** Given a graph $G=(V,E)$, σ is a neighbourhood function from V to 2^V.

Example 2 One obvious definition of σ for an undirected graph (V, E) is $\sigma(v)=\{v_i : v_i \in V, (v, v_i) \in E\}$. For the graph in Example 1, $\sigma(1)=\{2,6\}, \sigma(2)=\{1,3\}$.

For the GNNs in this paper, we will need labelled undirected graphs.

Definition 3 **(Graph Labellings)** Let $\mathcal{V}$ be a set of vertex labels and $\mathcal{E}$ be a set of edge labels. Then a vertex-labelling of a graph $G=(V,E)$ is a function $\psi : V \rightarrow 2^{\mathcal{V}}$ and an edge-labelling is a function $\epsilon : E \rightarrow 2^{\mathcal{E}}$.[3]

Example 3 The vertex labels of the graph given in Example 1 can be the atom-types (Carbon, C), and edge labels can be the bond-types (single bond: 1, double bond: 2). The label for the vertex 1 is $\psi(1)=\cdots=\psi(6)=\{C\}$. The labelling for the edges are $\epsilon((1,2))=\epsilon((2,1))=\{2\}, \epsilon((2,3))=\epsilon((3,2))=\{1\}$ and so on.

Although not evident in this example, vertex- and edge-labels can have more than one element (hence the mapping to $2^{\mathcal{V}}$ and $2^{\mathcal{E}}$). This will be necessary later.

We will use the term *graph* interchangeably to denote the tuple (V, E) or the tuple $(V,E,\sigma,\psi,\epsilon)$. We are interested here in classifying graphs. That is, given a set of class labels $\mathcal{Y}$, we want to construct a function that maps a graph of the form $(V,E,\sigma,\psi,\epsilon)$ to $\mathcal{Y}$. A GNN is one such function that employs 2 higher-order functions.

Definition 4 **(Relabel)** Given a graph $(V,E,\sigma,\psi,\epsilon)$. Let *Relabel* be a function that returns a graph $(V,E,\sigma,\psi',\epsilon')$, where the functions ψ' and ϵ' may be different to ψ and ϵ.

A vectorisation function is used to map a graph as a real-valued vector.

[2] Henceforth, by "graph" we will mean an undirected graph.

[3] We do not commit here to any specific data structure that should be used to implement the label set. This could be, for example, a Boolean-valued array of size $|\mathcal{V}|$.

Definition 5 **(Vectorise)** Let $\mathcal{G}$ denote the set of graph-tuples of the form $(V, E, \sigma, \psi, \epsilon)$. A vectorisation of the graph-tuple is the result of applying a function $Vec : \mathcal{G} \to \Re^d$ $(d \geq 1)$.

A GNN is the composition of these functions, and some prediction function as implemented by a neural network.

Definition 6 **(GNN)** Let $NN : \Re^d \to \mathcal{Y}$ denote a neural network that maps a real-valued vector to a set of class labels. Given a $G = (V, E, \sigma, \psi, \epsilon)$, $GNN(G) = NN(Vec(Relabel(G)))$.

Variations of GNNs result from changing the definitions of *NN*, *Vec* and *Relabel*. Many different definitions of the *Relabel* function have been proposed recently. We defer the specific details of the GNN variants used here to Section 2.2.

2.1 Encoding *n*-ary relations

GNNs, as we have described them so far, deal with node- and edge-labels in an undirected graph, in which edges are sets of vertex-pairs. That is, the edges represent a symmetric binary relation. However, for many real-world problems—including the ones considered in this paper—we have access to domain-knowledge which relate more than just pairs of vertices. For example, if a molecule is represented as a graph (with atoms as vertices, and an edge denoting a bond between a pair of vertices), then a benzene-ring is a relation amongst 6 distinct vertices, with some specific constraints on the vertices and edges. Here, we will consider domain-knowledge to be a set of relations, each of which can be expressed as a hypergraph.

Definition 7 **(Hypergraphs)** A hypergraph H is the pair (V, E'), where V is a set of vertices and E' is a non-empty subset of 2^V. Each element of E' is called a hyperedge.

Example 4 A hypergraph of the molecular graph given in Example 1 can be $H = (\{1, 2, 3, 4, 5, 6\}, \{\{1, 2\}, \{3, 4, 5, 6\}, \{2, 4, 5\}, \{1, 2, 3, 4, 5, 6\}\})$.

We note that since hyperedges are sets, there is no distinction between permutations of vertices in a hyperedge. So, as defined here, we will take hyperedges as being undirected. Hypergraph labellings can be defined similarly as before, using a pair of functions for vertex- and edge-labels. We will reuse the notation ψ and ϵ for these functions, with annotations to clarify what is meant. The neighboorhood relation σ is left unspecified here (one obvious definition is $\sigma(v_i) = \{v_j : h \in E', \{v_i, v_j\} \subseteq h\}$). In this paper, we are interested in n-ary relations that can be expressed as hypergraphs.

Definition 8 **(n-ary relation as a labelled hypergraph)** A n-ary relation R defined over vertices of a graph $G = (V, E)$ is a hypergraph $H = (V, E')$, and every hyperedge $h \in E'$ has n elements from V. We will denote this as $R(G) = H$. Let ψ_G denote a vertex-labelling over G and R/n denote the predicate-symbol for R. With some abuse of notation, the vertex-labelling function for $R(G) = H = (V, E')$ is as follows:

$$\psi_H(v) = \begin{cases} \psi_G(v) \cup \{R/n\} & \text{if } \exists h \in E' s.t.\ v \in h \\ \emptyset & \text{otherwise} \end{cases}$$

and the hyperedge-labelling function is:

$$\epsilon_H(h) \;=\; \{R/n\} \quad (h \in E')$$

That is, the vertex-labelling of a vertex v in the hypergraph H is a set containing the existing vertex-label of v in G augmented by the predicate-symbol R/n vertex-label.

Example 5 Consider a relation for a Benzene ring:

$$\begin{aligned} &Benzene(a_1, a_2, a_3, a_4, a_5, a_6) \leftarrow \\ &\qquad\qquad Cycle(a_1, a_2, a_3, a_4, a_5, a_6) \wedge \\ &\qquad\qquad Aromatic(a_1, a_2, a_3, a_4, a_5, a_6). \end{aligned}$$

One possible vertex-labelling is:

$$\psi_H(1) = \cdots = \psi_H(6) = \{C, Benzene/6\}$$

(here, C denotes “carbon”). A hyperedge-labelling may contain:

$$\epsilon_H(\{1,2,3,4,5,6\}) = \{Benzene/6\}$$

The extension to multiple relations, not all of the same arity, is straightforward.

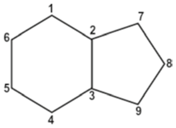
Definition 9 (Multiple relations as a labelled hypergraph) Let $R_1, \dots, R_k$ be relations defined on vertices of a graph $G = (V, E)$, s.t. $R_i(G) = (V, E_i{}')$. Then $\bigcup R_i(G)$ is the hypergraph $H = (V, E')$ where $E' = \bigcup E_i{}'$. The corresponding labelling functions are:

$$\psi_H(v) = \bigcup \psi_{H_i}(v)$$

and

$$\epsilon_H(v) = \bigcup \epsilon_{H_i}(v)$$

Example 6 In the molecular graph given below, there are two relations: *Benzene*/6 and *Pyrrole*/5.



One possible vertex-labelling for this graph is:

$$\begin{aligned} \psi_H(1) =& \psi_H(4) = \psi_H(5) = \psi_H(6) = \{C, Benzene/6\} \\ \psi_H(8) =& \psi_H(9) = \{C, Pyrrole/5\} \\ \psi_H(7) =& \{N, Pyrrole/5\} \\ \psi_H(2) =& \psi_H(3) = \{C, Benzene/6, Pyrrole/5\} \end{aligned}$$

and a hyperedge-labelling is:

$$\epsilon_H(\{1,2,3,4,5,6\}) = \{Benzene/6\}$$
$$\epsilon_H(\{2,7,8,9,3\}) = \{Pyrrole/5\}$$

In principle, provided we are able to define a neighbourhood function σ for hypergraphs, the definition of GNNs in Defn. 6 does not change. We would however like to use one of the standard GNN implementations described in the previous section, which restricts graphs with 2-vertex edges, and edge-labels to singleton sets. With some loss of information, we extract a suitable graph from a hypergraph.

Definition 10 (Vertex-enriched graphs) Let $G = (V, E)$ be a graph, with neighbourhood function σ, vertex-labelling function ψ, and edge-labelling function ϵ. Here, E is a subset of $V \times V$. Let $\mathcal{R} = \{R_1, \ldots, R_k\}$ be a set of relations defined on G, and $\bigcup R_i(G)$ be the hypergraph $H = (V, E')$ with vertex-labelling function ψ' as in Defn. 9. Then $G' = (V, E, \sigma, \psi', \epsilon)$ is called a vertex-enriched form of $G = (V, E, \sigma, \psi, \epsilon)$. We denote this by $VE(G, \mathcal{R}) = G'$.

Example 7 The molecular graph G for Example 6 is

$$G =(\{1,2,3,4,5,6,7,8,9\}, \{(1,2),(2,1),\cdots,(1,6),(6,1),(2,7),(7,2),\cdots,$$
$$(9,3)(3,9)\})$$

A vertex-labelling of G is:

$$\psi(1) = \cdots = \psi(6) = \psi(8) = \psi(9) = \{C\}$$
$$\psi(7) =\{N\}$$

The vertex-labelling of the vertex-enriched graph G', after the inclusion of the relations in Example 6 is:

$$\psi'(1) =\psi'(4) = \psi'(5) = \psi'(6) = \{C, Benzene/6\}$$
$$\psi'(8) =\psi'(9) = \{C, Pyrrole/5\}$$
$$\psi'(7) =\{N, Pyrrole/5\}$$
$$\psi'(2) =\psi'(3) = \{C, Benzene/6, Pyrrole/5\}$$

The edge-labelling and neighborhood functions do not change after relation-enrichment.

The vertex-enriched graph thus extends the vertex-labelling of a graph G, with the vertex-labels from the hypergraph H obtained from relations $R_1, \ldots, R_k$ defined on G. The resulting graph can be used directly by the implementations of GNNs described in the appendix. We note that the process of vertex-enrichment is a simplification of the full relational information available. For example, in the example above, if an atom (represented by a vertex in the molecular graph) is part of more than 1 benzene ring, then its vertex-enrichment will only contain a single entry for *Benzene*/6, indicating that it is part of 1 or more benzene rings.

Definition 11 (Vertex-enriched GNN) Let $G = (V, E, \sigma, \psi, \epsilon)$, and *Relabel*, *Vec* and *NN* be as before. Then, a Vertex Enriched GNN is $VEGNN(G) = NN(Vec(Relabel(VE(G, \mathcal{R}))))$.

2.2 Practical considerations

The GNN variants in this paper differ in the *Relabel* operation, based on the convolution procedure employed. In this work, we employ the following different convolution procedures:

1. Localised approximation to spectral graph convolution (Kipf and Welling 2017): This is a spectral method for graph convolution that uses convolutional aggregator. This is a simple and well-behaved layer-wise propagation rule for neural network models which operate directly on graphs.
2. Multi-scale graph convolution (Morris et al. 2019): This convolution method can perform convolution operations using multiple-sized neighbourhoods (the authors call this "higher order" graph convolution).
3. Graph convolution with attention (Veličković et al. 2018): This is a spatial method of graph convolution that uses an "attention" mechanism, that estimates the importance of vertices in the neighbourhood of a vertex.
4. Sample-and-aggregate graph convolution (Hamilton et al. 2017): Here the convolution procedure samples from a distribution that is constructed from feature-vectors of vertices in the neighbourhood of a vertex.
5. Graph convolution based on auto-regressive moving average (Bianchi et al. 2019): This is a convolution method that employs a polynomial function of the feature-vectors in the neighbourhood of a vertex.

The *Relabel* operation also includes a pooling step after each convolution operation. Additional details are in Appendix A. In all cases, we have used a fixed vectorisation function *Vec* that is based on a readout mechanism, and *NN* refers to a standard multilayer perceptron (MLP).

We now elaborate on three practical issues arising from the use of Vertex-Enriched GNNs:

1. The vertex-enriched graphs we obtain allow us to use standard forms of GNNs (see Procedure 1). However, this comes with the limitation that we only change the vertex-labellings. A GNN defined directly on hypergraphs would have access to more information than the vertex-enriched GNN, since the former would retain the edge-labelling on hyperedges, and can have a richer definition of the neighbourhood function. Recently, there have been some proposals of GNNs for hypergraphs (Feng et al. 2019; Jiang et al. 2019; Yadati et al. 2019). It is possible that these forms of GNNs may perform better than Vertex-Enriched GNNs. We expect that the results in Section 3.4 will act as baseline for such comparisons.
2. Procedure 1 requires identification of subgraphs of the original graph. That is: for every relation $R_i \in \mathcal{R}$, the corresponding hyperedge H_i is a subset of vertices $\{v_1, \dots, v_n\} \in V$, such that $(v_1, \dots, v_n) \in R_i$. This step requires the identification of all subsets of vertices of the graph constituting hyperedge as above. For a graph (V, E), this can, in the worst case require an examination of $\binom{|V|}{n}$ combinations. Therefore, for arbitrary sized graphs and subgraphs, this is computationally hard. In practice, we will be forced to impose bounds on the size of V_s and on the size of the subgraph.

3. We have not described how the relations in $\mathcal{R}$ themselves are obtained. There are two possibilities here. First, they are provided as prior information (*background knowledge* in ILP terminology). Secondly, the $\mathcal{R}$ provided as prior information can be augmented by relations constructed automatically (see Procedure 2). In this paper, the construction of new relations is done using an ILP engine, by adapting the usual clause-construction procedure (see Appendix B.1 for an ILP-based implementation of *LearnRels* in Procedure 2).[4]

Procedure 1: (**EnrichGraph**) Vertex-Enrichment of a graph G, given a set of relations $\mathcal{R}$. The new label of a vertex includes all the relations of which the vertex is part.

```
Data: Graph $G = (V, E, \sigma, \psi, \epsilon)$, a set of relations $\mathcal{R} = \{R_1, \ldots, R_k\}$
Result: Vertex-Enriched Graph, $G' = (V, E, \sigma, \psi', \epsilon)$
Let $\psi' := \psi$;
for $R_i \in \mathcal{R}$ do
    Let $R_i \subseteq V^n$;
    $\mathcal{H}_i = \{\{v_1, \ldots, v_n\} : (v_1, \ldots, v_n) \in R_i\}$;
    Let $V_s = \bigcup_{H_j \in \mathcal{H}_i} H_j$;
    for $v_j \in V_s$ do
        $\psi'(v_j) := \psi'(v_j) \cup \{R_i/n\}$;
    end
end
return $(V, E, \sigma, \psi', \epsilon)$;
```

Procedure 2: (**AugmentRels**) Augmentation of a set of relations $\mathcal{R}$ by learning new relations from data.

```
Data: A graph $G = (V, E, \sigma, \psi, \epsilon)$; a set pre-classified instances $E = \{(G_i, y_i) :$
      $G_i = (V_i, E_i, \sigma, \psi_i, \epsilon_i)$ and $y_i \in \mathcal{Y}\}$; a set of relations $\mathcal{R} = \{R_1, \ldots, R_k\}$; and a
      bound $n$ on the number of new relations
Result: A vertex-enriched graph $G' = (V, E, \sigma, \psi', \epsilon)$ obtained from an augmentation
        of $\mathcal{R}$ by at most $n$ new relations obtained using $E$
Let $\mathcal{R}' = LearnRels(\mathcal{R}, E, n)$;
return $EnrichGraph(G, \mathcal{R} \cup \mathcal{R}')$;
```

3 Empirical evaluation

3.1 Aims

Our aims in this paper is to investigate the incorporation of background knowledge by GNNs. Specifically, using the term Vertex-Enriched GNNs (VEGNNs) to denote the inclusion of relations into GNNs (See Procedure 1), the experiments attempt to answer to the following questions:

[4] Usually, clauses constructed by an ILP engine are either used as part of a hypothesis, or as features to construct a Boolean-vector representation of the data ("propositionalisation"). Here, the clauses are not used in either of these roles, but as relations that augment the prior knowledge available to the GNN.

1. How do VEGNNs perform against standard GNNs? This compares GNNs with and without the inclusion of domain-knowledge.
2. Can the performance of VEGNNs be improved by using symbolic learner with access to the same domain-knowledge? This tests whether the computational machinery of a GNN is sufficient to construct (representations of) the high-level relationships needed for good prediction.

3.2 Materials

3.2.1 Data

The datasets are classification problems arising in the field of drug-discovery. We have evaluated our GNNs on 73 real-world binary classification datasets. Each dataset represents an extensive drug evaluation effort at the National Cancer Institute (NCI)[5]. The datasets represent experimentally determined effectiveness of anti-cancer activity of a compound against a number of cell lines (Marx et al. 2003). (Table 1)The datasets correspond to the concentration parameter GI50, which is the concentration that results in 50% growth inhibition. Some of the datasets have been used in various data mining studies such as in a study involving the use of graph kernels in machine learning (Ralaivola et al. 2005).

3.2.2 Background knowledge

The initial version of the background knowledge in this paper here was used in Van Craenenbroeck et al. (2002), (Ando et al. 2006). It is a collection of logic programs defining almost 100 relations for various functional groups and ring structures in a chemical compound.[6] The background knowledge consists of multiple hierarchies. However, we modified some of the predicate definitions to avoid redundant computation and for tractability to trade-off completeness for efficiency. For proprietary reasons, we are only able to show the results of using the definitions, which are functional groups represented as `functional_group(CompoundID, Atom, Length, Type)` and rings described as `ring(CompoundID, RingID, Atoms, Length, Type)`. For efficiency, we have restricted the definition of the ring relation to produce rings of maximum length 8. The first use of this new version of the background knowledge is reported in (Dash et al. 2018) where we had also defined three higher level relations to infer the presence of composite structures from the presence of functional groups and rings in a compound. These are: the presence of fused rings, connected rings and substructures. These relations are defined below.

`has_struc(CompoundId, Atoms, Length, Struc)`This relation is *TRUE* if a compound identified by `CompoundId` contains a structure `Struc` of length `Length` containing a set of atoms in `Atoms`

`fused(CompoundId, Struc1, Atoms1, Struc2, Atoms2)`This relation is *TRUE* if a compound identified by `CompoundId` contains a pair of fused structures `Struc1` and `Struc2` with `Atoms1` and `Atoms2` respectively (that is, there is at least 1 pair of common atoms).

`connected(CompoundId, Struc1, Atoms1, Struc2, Atoms2)`

[5] https://www.cancer.gov/

[6] The definitions used were originally developed for tackling industrial-strength problems by the biotechnology company PharmaDM.

Table 1 Summary of datasets (Total number of instances is 221306)

# of Datasets	Avg. # of Molecules per dataset (Graphs)	Avg. # of Atoms per molecule (Vertices)	Avg. # of Bonds per molecule (Edges)
73	3032	24	51

This relation is *TRUE* if a compound identified by `CompoundId` contains a pair structures `Struc1` and `Struc2` that with `Atoms1` and `Atoms2` respectively that are not fused but connected by a bond between an atom in `Struc1` and an atom in `Struc2`.

The level of abstraction in the background knowledge is shown in Fig. 1. The hierarchy available in functional groups and rings is shown in Fig. 2 and Fig. 3.

3.2.3 Algorithms and machines

The data used for this work and the set of symbolic relations ($\mathcal{R}$) described in Section 3.2.2 are written as Prolog facts. For generating the additional set of ILP relations ($\mathcal{R}'$), we use Aleph (Srinivasan 2001) that takes the data and the background-knowledge as input. This additional set of relations $\mathcal{R}'$ further augments the existing relations in $\mathcal{R}$ for our *VEGNN′* studies. A logic program extracts a set of vertices in a graph for which any symbolic relation R_i ($\in \mathcal{R}$ or $\in \mathcal{R}'$) is *TRUE*. We use YAP compiler for execution of this logic program.

The GNN variants used here are described in Appendix A. All the experiments are conducted in Python environment. The GNN models have been implemented by using the PyTorch Geometric library (Fey and Lenssen 2019), which is a geometric deep learning extension for PyTorch (Paszke et al. 2019) and it provides graph pre-processing routines and makes the definition of graph convolution easier to implement.

For all the experiments, we use a machine with Ubuntu (16.04 LTS) operating system, and hardware configuration such as: 64GB of main memory, 16-core Intel Xeon processor, a NVIDIA P4000 graphics processor with 8GB of video memory.

3.3 Method

In all experiments, we refer to GNN variants as $GNN_{1,\ldots,5}$. The corresponding vertex-enriched versions are $VEGNN_{1,\ldots,5}$. The GNN variants have 1 hyper-parameter that determines the structure of the GNN (see Appendix A). We will denote this by m and assume that it takes values from a fixed-set of values M.

Experiment 1: GNNs vs. VEGNNs

For constructing the VEGNNs, we assume that we have access to a set of domain relations $\mathcal{R}$. The method used is as follows.

For each dataset D:

1. Let *Tr*, *Val*, *Te* denote a train-validation-test split of the data D
2. For each of $GNN_{1,\ldots,5}$ and $VEGNN_{1,\ldots,5}$:

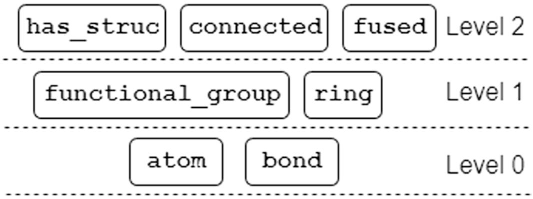


Fig. 1 Levels of abstraction in the background knowledge (Dash et al. 2018)

(a) Find the best value $m^* \in M$ using the performance on Tr and Val
(b) Record the predictive performance on *Te* of the model constructed using m^*

3. Compare the performance of GNN_i against that of $VEGNN_i$ ($i = 1, \ldots, 5$).

The following additional details are relevant:

- The relations in $\mathcal{R}$ are those described in Section 3.2.2.
- In our implementation, we use three graph convolution blocks and three pooling blocks interleaving each other.
- The convolution blocks can be of one of the five convolution variants listed in Section 2.2. Due to the large-scale experimentation (number of datasets, number of GNN variants), the various hyperparameters in convolution blocks are set to default values in PyTorch Geometric library.
- The graph pooling block uses self-attention pooling (Lee et al. 2019) with pooling ratio of 0.5. We use a hierarchical pooling architecture that uses the readout mechanism proposed by Cangea et al. (2018). The readout block aggregates node features to produce a fixed size intermediate representation for the graph. The final fixed-size representation for the graph is obtained by element-wise addition ($\oplus$) of the three readout representations.
- The final representation is then fed as input to a 3-layered MLP. We use a dropout layer with fixed dropout rate of 0.5 after first layer of MLP. The loss function is negative log-likelihood between the targets and the predictions from the model. Further detail on the GNN architectures is provided in Appendix A.4.
- We select amongst two possible values of the structure hyperparameter m (8 and 128), corresponding to small and large amounts of convolution in the convolutional-layers of the GNNs and VEGNNs;
- We use (Adam Kingma and Ba 2014) optimiser for training the GNNs ($GNN_{1,\ldots,5}$) and VEGNNs ($VEGNN_{1,\ldots,5}$). The learning rate is 0.0005, weight decay parameter is 0.0001, momentum factors are the default values of $\beta_{1,2} = (0.9, 0.999)$.
- Maximum number of training epochs is 1000. The batch size is 128.
- We use an early-stopping mechanism (Prechelt 1998) to obtain the optimal model after training that can be used for evaluation on *Te*. The patience period for early stopping is 50.
- Comparison of performance is done using the Wilcoxon signed-rank test, using the standard implementation within MATLAB (R218b).

Experiment 2: VEGNNs with ILP-constructed relations

Given a set of generic relations $\mathcal{R}$, and some data, a VEGNN should, in principle, be able to construct new (domain-specific) relations across its internal layers. That is, it may

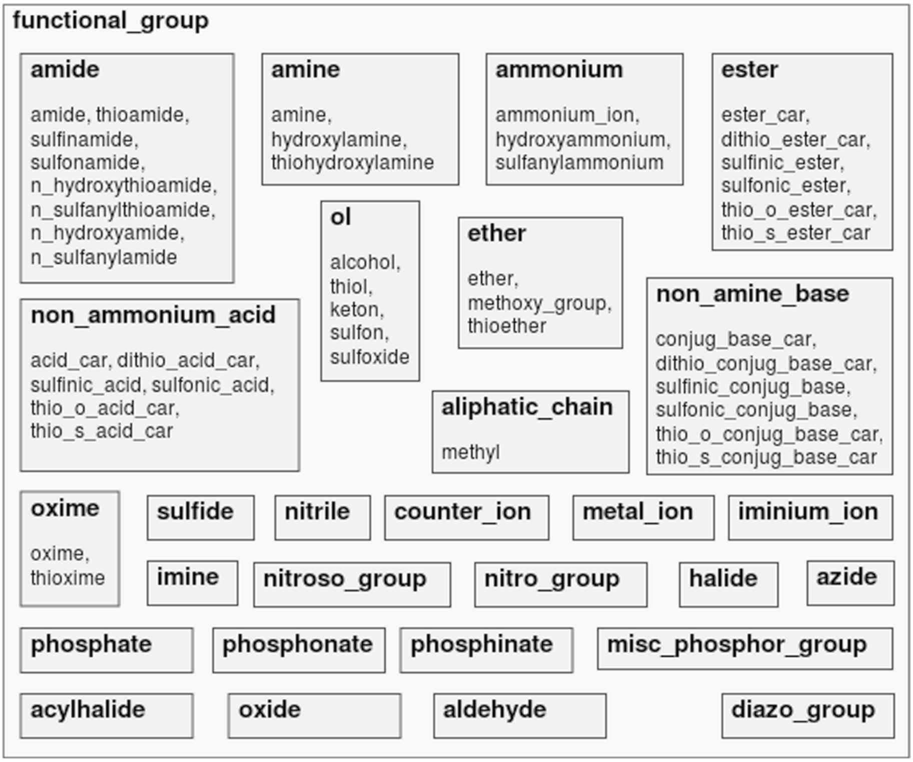

functional_group
amide
amide, thioamide, sulfinamide, sulfonamide, n_hydroxythioamide, n_sulfanylthioamide, n_hydroxyamide, n_sulfanylamide
amine
amine, hydroxylamine, thiohydroxylamine
ammonium
ammonium_ion, hydroxyammonium, sulfanylammonium
ester
ester_car, dithio_ester_car, sulfinic_ester, sulfonic_ester, thio_o_ester_car, thio_s_ester_car
ol
alcohol, thiol, keton, sulfon, sulfoxide
ether
ether, methoxy_group, thioether
non_ammonium_acid
acid_car, dithio_acid_car, sulfinic_acid, sulfonic_acid, thio_o_acid_car, thio_s_acid_car
aliphatic_chain
methyl
non_amine_base
conjug_base_car, dithio_conjug_base_car, sulfinic_conjug_base, sulfonic_conjug_base, thio_o_conjug_base_car, thio_s_conjug_base_car
oxime
oxime, thioxime
sulfide
nitrile
counter_ion
metal_ion
iminium_ion
imine
nitroso_group
nitro_group
halide
azide
phosphate
phosphonate
phosphinate
misc_phosphor_group
acylhalide
oxide
aldehyde
diazo_group

Fig. 2 Functional group hierarchy

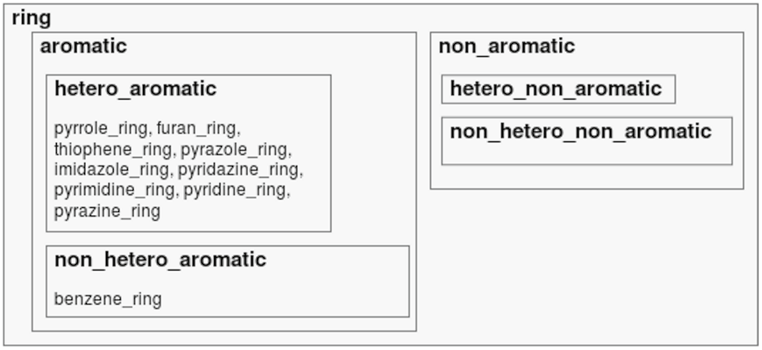

ring
aromatic
hetero_aromatic
pyrrole_ring, furan_ring, thiophene_ring, pyrazole_ring, imidazole_ring, pyridazine_ring, pyrimidine_ring, pyridine_ring, pyrazine_ring
non_hetero_aromatic
benzene_ring
non_aromatic
hetero_non_aromatic
non_hetero_non_aromatic

Fig. 3 Ring hierarchy

not be necessary to provide a VEGNN with anything more than $\mathcal{R}$. In this experiment, we investigate the extent to which this holds in practice, by evaluating the effects of augmenting $\mathcal{R}$ with higher-level relations learned by ILP. The ILP procedure used to obtain these relations has been described elsewhere (see Procedure 2). Our method is as follows.

For each dataset D:

1. Let Tr, Val, Te denote the train-validation-test split of the data D
2. Let $\mathcal{R}'$ denote a set of new relations obtained using an ILP engine with access to $\mathcal{R}$ and $Tr \cup Val$
3. Let $VEGNN_{1,\dots,5}$ denote the VEGNNs obtained with $\mathcal{R}$ and $VEGNN'_{1,\dots,5}$ denote the VEGNNs with $\mathcal{R} \cup \mathcal{R}'$.
   For each of $VEGNN_{1,\dots,5}$ and $VEGNN'_{1,\dots,5}$:
   (a) Find the best value m^* for the structure hyperparameter m, using Tr and Val
   (b) Record the predictive performance on Te of the model constructed using m^*
4. Compare the performance of $VEGNN_i$ against that of $VEGNN'_i$ $(i = 1, \dots, 5)$.

The following additional details are relevant:

- The relations in $\mathcal{R}$ are those described in Section 3.2.
- The construction of the *VEGNN*s is as in Experiment 1.
- The relations in $\mathcal{R}'$ are obtained using the ILP engine Aleph (Srinivasan 2001) with hide-and-seek sampling (Dash et al. 2019).
- We repeat the comparisons for $|\mathcal{R}'| = 100$, $|\mathcal{R}'| = 500$, and $|\mathcal{R}'| = 1000$.
- ILP-constructed relations can be complex, and involve several vertices. To ensure tractability, we restrict the computation to detecting a single hyperedge (and not all hyperedges) corresponding to the ILP-constructed relation. This results in a loss of information.
- As in Experiment 1, comparisons will be in the form of a Wilcoxon signed-rank test, implemented within MATLAB (R2018b).

3.4 Results

The main results from the experiments are shown qualitatively in Fig. 4. The principal findings from the tabulations are these: (a) Inclusion of domain-knowledge into GNNs (that is, the use of vertex-enriched GNNs) results in an improvement in predictive accuracy for all variants of GNN; and (b) The performance of vertex-enriched GNNs can be improved further by augmenting the domain-relations with additional relations constructed by an ILP engine.

We now examine the results in more detail: From Fig. 4, it is evident that the performance of graph-based networks improves with the inclusion of domain-knowledge. A quantitative tabulation of wins, losses and draws is in Table. 2. These results provide sufficient grounds to answer positively the primary research question addressed in this paper, namely: do GNNs benefit from the inclusion of domain-knowledge?

Assuming that it is useful to provide a GNN with domain-knowledge, we can then ask: are vertex-enriched GNNs sufficiently powerful to compute automatically any additional information needed for high predictive performance? The results in Fig. 4 suggest that the answer to this is "no", since it appears that the inclusion of ILP-constructed relations makes a significant difference. To understand this better, we tabulate quantitative differences obtained as the number of ILP relations added is increased. This is shown in

Table. 3. The plot in Fig. 4 uses 1000 ILP-relations (the corresponding quantitative differences are the last column in Table. 3).

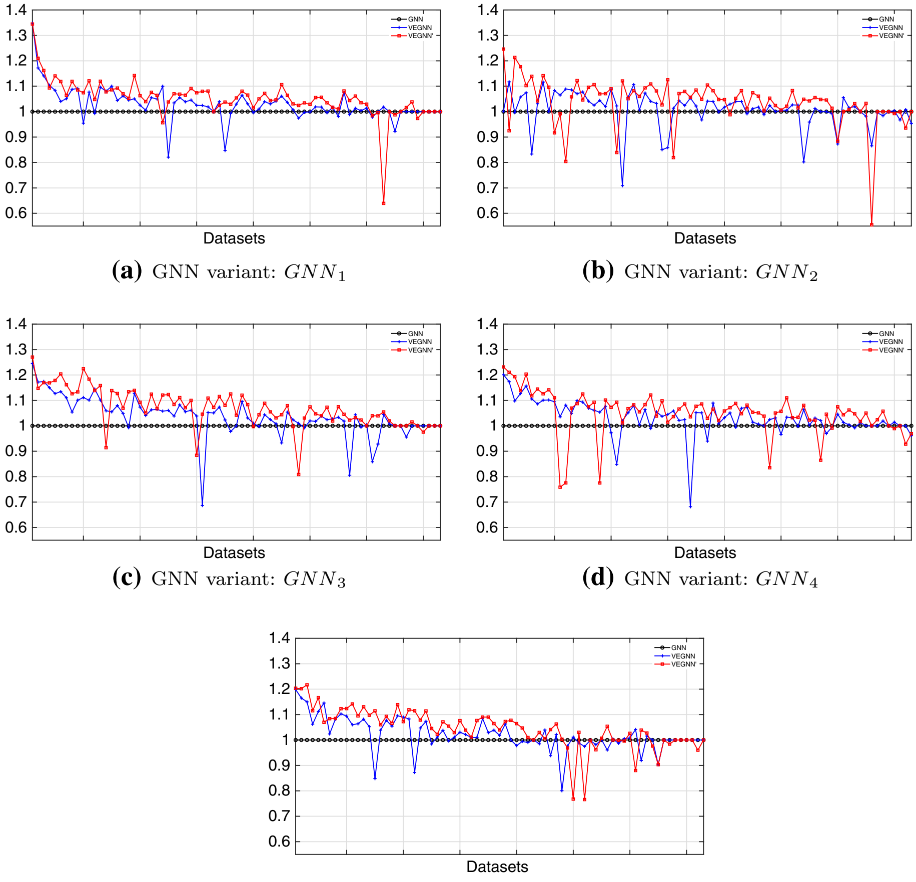
Since the inclusion of even small numbers of ILP relations (100) seems to improve performance of the VEGNN, it would appear that the internal representations within a VEGNN are of limited expressivity when compared to those constructed by ILP. In turn, the complete tabulation suggests that a hybrid VEGNN-ILP learner is very likely to be better than just a VEGNN learner (and in turn, a GNN learner).

We note that vertex-enrichment is only a vertex-related operation. It is relevant to ask if there are any edge-related operations associated with the addition of domain-relations. Since these relations result in hyperedges, a natural edge-operation is one of *clique-expansion* (Zhou et al. 2007) of the domain-relations. That is, the original graph is transformed to a new graph by the inclusion of all pairwise edges between vertices in hyperedges entailed by the relations. We have investigated this, but for reasons of space, do not include the results here. A summary of the effect of clique-expansion is: (a) By itself, clique-expansion of domain-relations is not helpful; (b) Clique-expansion, in combination with vertex-enrichment does not yield any clear advantage over vertex-enrichment alone across the GNN variants.

4 Related work

GNN-like models were first proposed in Sperduti and Starita (1997), (Baskin et al. 1997). In these studies, the features from the graph data was extracted using neural networks. Gori et al. (2005) and Scarselli et al. (2008) proposed new graph-based learning methods that used recursive aggregation of information. They called these models ‘graph neural networks (GNNs)’. The major boost to the field of GNNs followed the introduction of graph convolution (Kipf and Welling 2017) and the notion of graph embedding (Cui et al. 2018; Zhang et al. 2018). Many such embedding methods are based on iterative processing of the neighborhood information of any vertex. One such vertex embedding method was formulated by generalising the convolution operation to graphs. The convolution operation computes “hidden” states (essentially vector-representations) of the vertices in the graph. There are a wide variety of convolution-based GNNs most of which are classified into spectral- or non-spectral (spatial) approaches. Two methodical and comprehensive surveys over a series of variants of graph neural networks can be found in (Zhou et al. 2018) and (Wu et al. 2020). We have already seen that for practical problems the data cannot effectively be modelled by pairwise associations. Methods have been proposed to define convolutions for higher-order graphs or hypergraphs (Feng et al. 2019; Jiang et al. 2019; Yadati et al. 2019), although none of these have considered the problem of inclusion of domain-knowledge. To the extent that we consider a vertex-enriched graph to be a result of a hypergraph representation of the data, the work proposed in this paper loosely falls under the category of Hypergraph-based neural networks.

Notwithstanding the convolution operation used in GNNs, one drawback that has been identified is that the representations learned by them could be poor if the amount of training data (number of graphs) is small, which would lead to poor generalisation (Xu et al. 2019). The usual solution to this problem is to overcome data scarcity by the use of prior knowledge, a feature that is at the heart of Inductive Logic Programming. In almost all applications of ILP to date, the use of prior or *b*ackground knowledge is central (see Muggleton et al. 2012). In contrast, the position taken in the neural-network literature, especially



(a) GNN variant: GNN_1

(b) GNN variant: GNN_2

(c) GNN variant: GNN_3

(d) GNN variant: GNN_4

(e) GNN variant: GNN_5

Fig. 4 Qualitative comparison of graph-based neural networks. Here GNN refers to the performance of the graph-based neural network without domain relations; VEGNN refers to the performance of the network vertex-enriched with generic domain relations shown in Section 3.2.2; and VEGNN′ refers to the performance of the network vertex-enriched with the generic domain-relations and domain-specific relations constructed by an ILP engine. Performance refers to predictive (holdout-set) accuracy, and all performances are normalised against that of the GNN. Further, the compounds are arranged in order of increasing GNN performance: the apparent trend of high-to-low gains for VEGNN and VEGNN′ from left to right are artifacts of this ordering. No significance should also be attached to the line joining the data points: this is only for visual clarity

those dealing with networks with large numbers of hidden layers, is that provided sufficient data are available, representations of relevant domain-concepts can be computed automatically from data. But when data are scarce, this assumption breaks down. The area of neuro-symbolic modelling (Besold et al. 2017) has been concerned with ways of combining symbolic and neural learning. A simple way of doing this has been studied under the category of "propositionalisation" in ILP (Lavrač et al. 1991; Kramer et al. 2001; Krogel et al. 2003; França et al. 2014, 2015). Although, propositionalisation approaches have been successfully applied to various problems but are still considered as ad hoc approaches. These approaches are studied in the larger context of macro-operators (Castillo and Wrobel

Table 2 Quantitative comparison of GNN performance. Here *GNN* refers to the graph-based neural network without domain-knowledge, and *VEGNN* refers to the network vertex-enriched with the generic domain-knowledge described in Section 3.2.2. The tabulations are the number of datasets on which *VEGNN* has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test

GNN Variant	Accuracy (*VEGNN* vs. *GNN*) Higher/Lower/Equal (*p*-value)
GNN_1	48/14/11 (< 0.001)
GNN_2	48/19/6 (0.005)
GNN_3	53/11/9 (< 0.001)
GNN_4	54/12/7 (< 0.001)
GNN_5	43/19/11 (0.002)

Table 3 Quantitative comparison of performance after augmenting domain-relations with ILP-constructed relations. Here *VEGNN′* denotes the vertex-enriched GNN obtained after augmenting the generic domain relations ($\mathcal{R}$) with domain-specific relations constructed by an ILP engine ($\mathcal{R}'$); and *VEGNN* denotes the vertex-enriched GNN with $\mathcal{R}$. The tabulations are the number of datasets on which *VEGNN′* has higher, lower or equal predictive accuracy on a holdout-set. Statistical significance is assessed by the Wilcoxon signed-rank test

	Accuracy (*VEGNN′* vs. *VEGNN*)		
GNN	Higher/Lower/Equal (*p*-value)		
Variant	$\lvert\mathcal{R}'\rvert = 100$	$\lvert\mathcal{R}'\rvert = 500$	$\lvert\mathcal{R}'\rvert = 1000$
GNN_1	45/17/11 (< 0.001)	46/19/8 (< 0.001)	55/10/8 (< 0.001)
GNN_2	46/20/7 (< 0.001)	55/13/5 (< 0.001)	54/17/2 (< 0.001)
GNN_3	47/17/9 (< 0.001)	49/16/8 (< 0.001)	55/12/6 (< 0.001)
GNN_4	40/27/6 (0.055)	46/23/4 (0.013)	53/16/4 (< 0.001)
GNN_5	39/20/14 (0.026)	49/14/10 (< 0.001)	51/13/9 (< 0.001)

2002), which are approaches to improve the heuristic search in ILP systems and extract higher-level or meta-rules (Alphonse 2004). Pioneering work on the combination of neural-networks and symbolic features has been done by d'Avila (Garcez and Zaverucha 1999) and extended in (França et al. 2014, 2015). There are several studies that report that the relational features constructed using propositionalisation-based approach can substantially improve predictive performance of statistical machine learning models, see for example: (Ramakrishnan et al. 2007), (Saha et al. 2012). Recently, ILP-based feature-construction for deep multi-layer perceptrons [a special case of Deep Relational Machines, or DRMs (Lodhi 2013) was shown to yield surprisingly good results on the datasets used here, albeit with very large numbers of features (Dash et al. 2018, 2019). At the other end of the spectrum, methods are now being developed that include "neural" predicates (predicates whose definitions are implemented by neural networks) as part of the background knowledge available to a symbolic learner (De Raedt et al. 2019).

Domain-knowledge is often available as knowledge graphs (or semantic networks) rather than as a set of relations defined in logic. Knowledge graph embedding (Ding et al. 2018; Ziegler et al. 2017) is a technique that is mostly applied to construct a vector representation for the knowledge graph, which can then be *infused* into some form into a neural network. In recent reports, it is proposed that the latent representation learned by a neural network can be coupled with the representation of the knowledge graph that may improve the predictive performance of the neural network model (Gaur et al. 2019; Kursuncu et al. 2019).

5 Conclusions

Our focus in this paper has been on the use of graph-based neural networks (GNNs) on scientific data. Scientific understanding is largely an incremental process that builds on knowledge that is already known. It is natural therefore to expect that automatic techniques intended for scientific data analysis will similarly be able to utilise such knowledge. The results here clearly show the benefit of having mechanisms to incorporate domain-knowledge into GNNs. They also show the benefits of ILP as a mechanism for identifying relationships that appear not to be within the practical reach of the GNN variants we have considered. An ILP-purist could well ask: why then should we use GNNs at all? There are several reasons to persist, chief amongst which are reasons of implementation efficiency and widespread availability of packaged libraries. Assuming GNNs are useful, our goal has been to show that they can be more useful if they use domain-specific relations, and yet more so if they include results from an ILP engine.[7]

To the best of our knowledge, the experiments in this paper constitute some of the most extensive applications of GNNs to large-scale real-world scientific data. It has not been the focus of this paper to construct a GNN-based benchmark for the data, but to investigate the use of domain-knowledge. There is undoubtedly room in the future for comparative studies against other techniques that may or may not utilise the domain-knowledge available. More immediately, the process of vertex-enrichment can create very large vectors at each vertex (the result of a many-hot encoding of the relations in the vertex's label). We conjecture that this situation can be improved by performing some dimensionality-reduction at each vertex. A straightforward option is to include some form of auto-encoder at each vertex, before re-labelling. Vertex-enriched GNNs can probably be significantly improved by directly working with Hypergraph GNNs (HGNNs). In principle, HGNNs will have more information (like hyperedge labels). Will HGNNs also benefit from the use of ILP? We do not know the answer to this as yet.

Despite the recent empirical successes in various fields, recent studies highlight some of the theoretical limitations of GNNs. For instance, GNNs cannot distinguish between some pairs of graphs that are indistinguishable by the 1-WL test (Xu et al. 2019), (Morris et al. 2019), that is, a GNN with any parameter setting cannot distinguish two graphs unless the labels of the graphs are same. A recent study on GNNs (Barceló et al. 2020) has shown that the class of aggregate-combine GNNs cannot be logically more expressible than a fragment of two-variable first-order logic with counting quantifiers (Logic FOC2), which is a form of description logic. In a different report, various theoretical limitations of GNNs are studied, specifically, in terms of approximation ratios of combinatorial algorithms (Sato 2020). We have already indicated that the vertex-enrichment procedure described in this

[7] The use of ILP would seem to undermine the motivation just given for using GNNs. However, this is not so. First, once the ILP relations are constructed, the main modelling effort is still done using GNNs. Secondly, the construction of relations is task that can be implemented by a specialised library.

paper may not capture fully the relational information present in the data. We believe this limitation can be overcome by adopting a different form of graph representation, that is nevertheless still amenable to the use of GNNs. We intend to explore this as future work.

At the outset of this paper, we motivated the use of machine learning in developing an automated scientific assistant. While high predictive power is expected from an ML-based scientific assistant, it is not sufficient. It is evident that this paper's focus is on how prediction can improve by the inclusion of domain-knowledge. An Understandable explanation of the models constructed by GNNs remains a challenge.

A Graph neural networks

A.1 Implementation

In a graph $G = (V, E)$, let X_v denote a vector that represents the labelling of a vertex $v \in V$. This is called the feature vector of the vertex v. In a GNN, the *Relabel* function is implemented by a neighbourhood aggregation mechanism (Xu et al. 2019). It updates the representation of a vertex, h_v iteratively. That is, in kth iteration (or kth layer), the representation of a vertex v, $h_v^{(k)}$ can be computed using two procedures: AGGREGATE and COMBINE.

$$a_v^{(k)} = \mathsf{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} \,:\, u \in \mathcal{N}(v)\right\}\right), \tag{1}$$

$$h_v^{(k)} = \mathsf{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right) \tag{2}$$

where, $\mathcal{N}(v)$ denotes the set of vertices adjacent to v. Initially (at $k = 0$), $h_v^{(0)} = X_v$.

The *Vectorise* function constructs a vector representation of the entire graph. This step is carried out after the representations of all the vertices are relabelled by some iterations. The vectorised representation of the entire graph can be obtained using a READOUT function that aggregates vertex features from the final iteration ($k = K$):

$$h_G = \mathsf{READOUT}\left(\left\{h_v^{(K)} \mid v \in G\right\}\right) \tag{3}$$

There are different variants of AGGREGATE-COMBINE procedures available in the literature on GNNs. These are mostly implemented using the methods known as graph convolution and graph pooling (refer Zhou et al. 2018; Wu et al. 2020). The READOUT procedure is usually implemented using a global or hierarchical pooling operation. The convolution operations of various GNNs used in our work are briefly described in Appendix A.2. Further, we use an additional pooling layer called structural-attention pooling after each of the convolution layer. This is briefly described in Appendix A.3.

A.2 Graph convolutions

A.2.1 Variant 1

The first variant of GNN used in our work is based on spectral-based graph convolutional network proposed by Kipf and Welling (2017). It uses a layer-wise (or iteration-wise) propagation rule for a graph with N vertices as:

$$\mathbf{H}^{(k)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\Theta^{(k-1)}\right) \tag{4}$$

where, $H^{(k)} \in \mathbb{R}^{N\times D}$ denotes the matrix of vertex representations of length D, $\tilde{A} = A + I$ is the adjacency matrix representing an undirected graph G with added self-connections, $A \in \mathbb{R}^{N\times N}$ is the graph adjacency matrix, I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $\Theta^{(k-1)}$ is the iteration-specific trainable parameter matrix, $\sigma(\cdot)$ denotes the activation function e.g. $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$, $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{X}$ is the matrix of vertex feature vectors X_is.

A.2.2 Variant 2

The second variant is based on the graph neural network proposed by Morris et al. (2019) that passes messages directly between subgraph structures inside the graph. At iteration k, the feature representation of a vertex is computed by using

$$h_u^{(k)} = \sigma\left(h_u^{(k-1)} \cdot \Theta_1^{(k)} + \sum_{v\in\mathcal{N}(u)} h_v^{(k-1)} \cdot \Theta_2^{(k)}\right) \tag{5}$$

where, σ is a non-linear transfer function applied component wise to the function argument, Θs are the layer-specific learnable parameters of the network.

A.2.3 Variant 3

The third variant is an attention-based model, which is popularly known as Graph Attention Network (GAT) (Veličković et al. 2018). This network assumes that the contributions of neighboring vertices to the central vertex are not pre-determined which is the case in the Graph Convolutional Network (Kipf and Welling 2017). This adopts attention mechanisms to learn the relative weights between two connected vertices. The graph convolutional operation at iteration k is thereby defined as:

$$h_u^{(k)} = \sigma\left(\sum_{v\in\mathcal{N}(u)\cup u} \alpha_{uv}^{(k)}\Theta^{(k)}h_u^{(k-1)}\right) \tag{6}$$

where, $h_u^{(0)} = X_u$. The connective strength between the vertex u and its neighbor vertex v is called attention weight, which is defined as

$$\alpha_{uv}^{(k)} = \mathrm{softmax}\left(\mathrm{LeakyReLU}\left(a^{\mathsf{T}}\left[\Theta^{(k)}h_u^{(k-1)} \parallel \Theta^{(k)}h_v^{(k-1)}\right]\right)\right) \tag{7}$$

where, a is the set of learnable parameters of a single layer feed-forward neural network.

A.2.4 Variant 4

The fourth variant is called GraphSAGE and it is a framework for inductive representation learning on large graphs (Hamilton et al. 2017). It is done in two steps: local neighborhood sampling and then aggregation of generating the embeddings of the sampled nodes. GraphSAGE is used to generate low-dimensional vector representations for nodes, and is especially useful for graphs that have rich node attribute information. The following is an iterative update of the node embedding:

$$h_u^{(k)} = \sigma\left(h_u^{(k-1)} \cdot \Theta_1^{(k)} + \frac{1}{|\mathcal{N}(u)|} \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} \cdot \Theta_2^{(k)} \right) \tag{8}$$

where, σ is a non-linear transfer function applied component wise to the function argument, Θs are the layer-specific learnable parameters of the network.

A.2.5 Variant 5

This variant of GNN is inspred by the auto-regressive moving avarage (ARMA) filters that are considered to be more robust than polynomial filters (Bianchi et al. 2019). The ARMA graph convolutional operator is defined as follows:

$$\mathbf{H}^{(k)} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{H}_m^{(K)} \tag{9}$$

where, M is the number of parallel stacks, K is the number of layers; and $\mathbf{H}_m^{(K)}$ is recursively defined as

$$\mathbf{H}_m^{(k+1)} = \sigma\left(\hat{L}\mathbf{H}_m^{(k)}\Theta_2^{(k)} + \mathbf{H}^{(0)}\Theta_2^{(k)} \right) \tag{10}$$

where, $\hat{L} = I - L$ is the modified Laplacian. The Θ parameters are learnable parameters.

A.3 Graph pooling

Graph pooling methods apply downsampling mechanisms to graphs. In this work, we use a recently proposed graph pooling method based on self-attention (Lee et al. 2019). It uses graph convolution defined in Eq. (4) to obtain a self-attention score as given in Eq. 11 with the trainable parameter replaced by $\Theta_{att} \in \mathbb{R}^{N \times 1}$, which is a set of trainable parameters in the pooling layer.

$$Z = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \mathbf{X} \Theta_{att} \right) \tag{11}$$

Here, $\sigma(\cdot)$ is the activation function e.g. tanh.

A.4 Structure of the GNNs

The structure of the GNNs closely follows the structure used in (Lee et al. 2019). A schematic diagram of our implemented architecture is shown in Fig. 5. As shown in the diagram, the output of the hierarchical pooling is fed as input to a multilayer perceptron (MLP). So, the input layer of the MLP contains $2m$ units, followed by two hidden layers with m units and $\lfloor m/2 \rfloor$ units respectively. The activation function used in the hidden layers is `relu`. The output layer size is $|\mathcal{Y}|$ (in this work, 2) with `logsoftmax` activation.

B ILP Specifics

B.1 Extending domain-knowledge

We assume that a set of relations $\mathcal{R}$ are provided as part of the background knowledge B available to an ILP engine.[8] Given B and data E consisting of a set of positive and negative instances (here representing molecules with or without the property of interest), and ILP engine can construct new clauses defined in terms of the relations in $\mathcal{R}$. These clauses can be additionally be ordered in terms of some utility function (for example, a clause encoding a relation that holds for large number of positive instances may have a high utility). The so-called technique of ILP-based "propositionalisation", for example, identifies high-utility clauses (for example, see Ramakrishnan et al. 2007; Joshi et al. 2008; Dash et al. 2018). The procedure used to draw "new" relations using ILP-derived techniques is in Procedure 3.

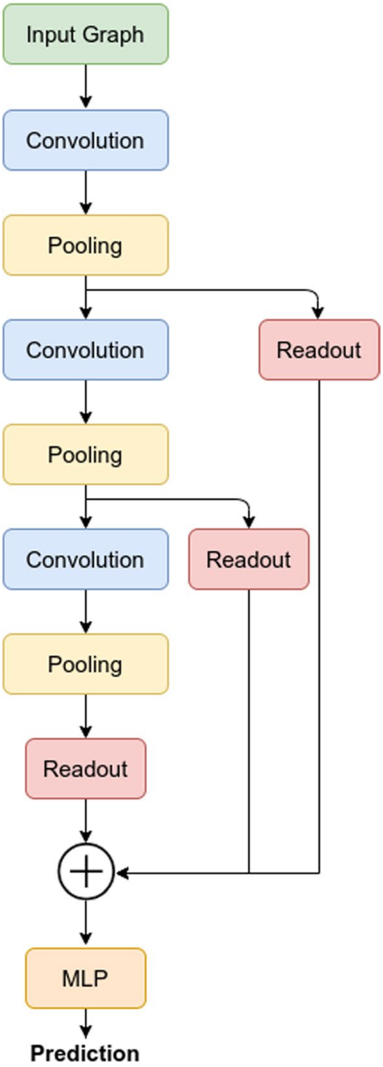
B.2 Input

We use the ILP engine Aleph to construct the most-specific rule above. Aleph requires the specification of a mode language, specifying the predicates in $\mathcal{R}$. The mode-language used for the experiments in the paper is given below:

```
:- modeb(*,bond(+mol,-atomid,-atomid,#atomtype,#atomtype,#bondtype)).
:- modeb(*,has_struc(+mol,-atomids,-length,#structype)).
:- modeb(*,connected(+mol,#structype,-atomids,#structype,-atomids)).
:- modeb(*,fused(+mol,#structype,-atomids,#structype,-atomids)).
```

[8] Besides $\mathcal{R}$, B will usually contain additional ILP-specific content like mode declarations (see Muggleton 1995, along with search constraints and ancillary predicates).

Fig. 5 Graph classification architecture used in this work. We perform our experiments with five different types of graph convolution methods, each resulting in a different kind of GNN architecture



Procedure 3: (**LearnRels**) Procedure to construct new relations using ILP. We assume the domain-knowledge consists of some relations $\mathcal{R} \in B$. The construction of $\bot$ is as described in [59], and $\succeq_\theta$ refers to Plotkin's θ-subsumption [61]. The redundancy test used is subsumption-equivalence. The distribution $\mathcal{D}_C$ is deliberately left unspecified here. In the experiments in the paper, $\mathcal{D}_C$ is either uniform (resulting in simple random construction of new relations), or a non-uniform selection based on clause-utility, as described in [28]. $NewR$ is a new relation name that does not occur in B; Cp denotes a conjunction of literals; $\mathbf{x}$ is shorthand for $x_1, x_2, \ldots, x_n$, the variables in Cp.

```
Data: Domain knowledge $B$, A set of examples $E$, and $MaxDraws$
Result: A set of relations $\mathcal{R}'$
$\mathcal{R}' := \emptyset$;
$draws := 0$;
$i := 1$;
$Drawn := \emptyset$;
while $draws \leq MaxDraws$ do
    Randomly draw an example $e_i \in E$ with replacement;
    Let $\bot(B, e_i)$ be the most specific rule that entails $e_i$, given $B$;
    Let $\mathcal{D}_C$ be a distribution over clauses;
    Draw a clause $C_i$ using $\mathcal{D}_C$ s.t. $C_i \succeq_\theta \bot_d(B, e_i)$;
    if $C_i$ is not redundant given $Drawn$ then
        Let $C_i = (Class((\mathbf{x}, c) \leftarrow Cp_i(\mathbf{x}))$;
        Let $R_i = (NewR(\mathbf{x}) \leftarrow Cp_i(\mathbf{x}))$;
        $Drawn := Drawn \cup \{C_i\}$;
        $\mathcal{R}' = \mathcal{R}' \cup \{R_i\}$;
        increment $i$;
    end
    increment $draws$;
end
return $\mathcal{R}'$;
```

The '#'-ed arguments in the mode declaration refers to type, that is, `#atomtype` refers to the type of atom, `#bondtype` refers to the type of bond, and `#structype` refers to the type of the structure (functional group or ring) associated with the molecule.

Each data instance (a molecule) is represented by a set of ground facts of the following kind:

```
bond(m1,27,24,o2,car,1).
...
```

Here `bond(m1,27,24,o2,car,1)` denotes that in instance `m1` there is an oxygen atom (id 27), and a carbon atom (id 24) connected by a single bond (`car` denotes a carbon atom in an aromatic ring).

Given the molecular structure additional facts like `functional_group/4` and `ring/4` are pre-computed for efficiency using the generic relations in $\mathcal{R}$ (which contain the symbolic definitions of benzene rings, oxide groups, *etc.*). This results in facts like the following:

```
functional_group(m1,[27],1,oxide).
ring(m1,[25,28,30,29,26,23],6,benzene_ring).
...
```

We note that these predicates result in a *reification* of the predicates in $\mathcal{R}$ (that is, the predicate symbols are converted to terms). The predicates has_struc/4, connected/5 and fused/5 are defined over these predicates. For example (in Prolog format):

```
 has_struc(Mol,Atoms,Length,Type):-
     ring(Mol,Atoms,Length,Type).
has_struc(Mol,Atoms,Length,Type):-
     functional_group(Mol,Atoms,Length,Type).
...
```

We reiterate that these predicates are defined directly on the relations in $\mathcal{R}$: the use of functional_group/4 and ring/4 is for compactness and efficiency.

B.3 Output

Given the mode language, and data consisting of the molecular structure, the ILP engine finds clauses like these (shown as Prolog clauses):

```
class(A,pos):-
    has_struc(A,D,E,ester_car),
    bond(A,F,G,c1,c1,3).

class(A,pos):-
    connected(A,benzene_ring,D,benzene_ring,E),
    connected(A,keton,F,non_hetero_non_aromatic,G).

class(A,pos):-
    fused(A,benzene_ring,D,imidazole_ring,E),
    connected(A,oxide,F,oxide,G).
...
```

Each such clause is converted to an *n*-ary relation using the steps in Procedure 2.

Acknowledgements This work is supported by DST-SERB Grant EMR/2016/002766, Government of India. The second author is a Visiting Professorial Fellow at School of CSE, UNSW Sydney. We sincerely thank Ing. Gustav Šourek, Czech Technical University, Prague for providing the dataset information; and researchers at the DTAI, University of Leuven, for suggestions on how to use the background knowledge within DMAX. We also thank Dr. Oghenejokpeme I. Orhobor and Professsor Ross D. King for providing us with initial set of background-knowledge definitions.

Data Availability Statement Data, background-knowledge and codes used in our experiments are available at: https://github.com/tirtharajdash/VEGNN.

References

Alphonse, É. (2004). Macro-operators revisited in inductive logic programming. In International Conference on Inductive Logic Programming. pp. 8–25. Springer

Ando, H. Y., Dehaspe, L., Luyten, W., Van Craenenbroeck, E., Vandecasteele, H., & Van Meervelt, L. (2006). Discovering h-bonding rules in crystals with inductive logic programming. Molecular pharmaceutics, 3(6), 665–674.

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J., & Silva, J. P. (2020). The logical expressiveness of graph neural networks. In International Conference on Learning Representations, https://openreview.net/forum?id=r1lZ7AEKvB

Baskin, I. I., Palyulin, V. A., & Zefirov, N. S. (1997). A neural device for searching direct correlations between structures and properties of chemical compounds. Journal of Chemical Information and Computer Sciences, 37(4), 715–721.

Besold, T. R., Garcez, A. d., Bader, S., Bowman, H., Domingos, P., Hitzler, P., Kühnberger, K. U., Lamb, L. C., Lowd, D., & Lima, P.M.V., et al. (2017). Neural-symbolic learning and reasoning: A survey and interpretation. arXiv preprint arXiv:1711.03902

Bianchi, F. M., Grattarola, D., Alippi, C., & Livi, L. (2019). Graph neural networks with convolutional arma filters. arXiv preprint arXiv:1901.01343

Cangea, C., Veličković, P., Jovanović, N., Kipf, T., & Liò, P. (2018). Towards sparse hierarchical graph classifiers. arXiv preprint arXiv:1811.01287

Castillo, L.P., & Wrobel, S. (2002). Macro-operators in multirelational learning: a search-space reduction technique. In European Conference on Machine Learning. pp. 357–368. Springer

Cui, P., Wang, X., Pei, J., & Zhu, W. (2018). A survey on network embedding. IEEE Transactions on Knowledge and Data Engineering, 31(5), 833–852.

d'Avila Garcez, A. S., & Zaverucha, G. (1999). The connectionist inductive learning and logic programming system. Appl. Intell., 11(1), 59–77.

Dash, T., Srinivasan, A., Vig, L., Orhobor, O. I., & King, R. D. (2018). Large-scale assessment of deep relational machines. In International Conference on Inductive Logic Programming. pp. 22–37. Springer

Dash, T., Srinivasan, A., Joshi, R.S., & Baskar, A. (2019). Discrete stochastic search and its application to feature-selection for deep relational machines. In International Conference on Artificial Neural Networks. pp. 29–45. Springer

De Raedt, L., Manhaeve, R., Dumancic, S., Demeester, T., & Kimmig, A. (2019). Neuro-symbolic= neural+ logical+ probabilistic. In NeSy'19@ IJCAI, the 14th International Workshop on Neural-Symbolic Learning and Reasoning. pp. 1–4

Ding, B., Wang, Q., Wang, B., & Guo, L. (2018). Improving knowledge graph embedding using simple constraints. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 110–121. Association for Computational Linguistics, Melbourne, Australia, https://www.aclweb.org/anthology/P18-1011

Faruquie, T. A., Srinivasan, A., & King, R. D. (2012). Topic models with relational features for drug design. In International conference on inductive logic programming. pp. 45–57. Springer.

Feng, Y., You, H., Zhang, Z., Ji, R., & Gao, Y. (2019). Hypergraph neural networks. Proceedings of the AAAI Conference on Artificial Intelligence., 33, 3558–3565.

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In ICLR Workshop on Representation Learning on Graphs and Manifolds

França, M. .V., Zaverucha, G., & Garcez, A. S. d. (2014). Fast relational learning using bottom clause propositionalization with artificial neural networks. Machine learning, 94(1), 81–104.

França, M.V.M., Zaverucha, G., & Garcez, A. S. d. (2015). Neural relational learning through semi-propositionalization of bottom clauses. In 2015 AAAI Spring Symposium Series

França, M.V.M., d'Avila Garcez, A. S., & Zaverucha, G. (2015). Relational knowledge extraction from neural networks. In Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches co-located with the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015), Montreal, Canada, December 11-12, 2015.

Gaur, M., Kursuncu, U., & Wickramarachchi, R. (2019). Shades of knowledge-infused learning for enhancing deep learning. IEEE Internet Computing, 23(6), 54–63.

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. vol. 2, pp. 729–734 vol. 2

Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In Advances in neural information processing systems. pp. 1024–1034

Jiang, J., Wei, Y., Feng, Y., Cao, J., & Gao, Y. (2019). Dynamic hypergraph neural networks. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI). pp. 2635–2641

Joshi, S., Ramakrishnan, G., & Srinivasan, A. (2008). Feature construction using theory-guided sampling and randomised search. In International Conference on Inductive Logic Programming. pp. 140–157. Springer

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

King, R. D., Muggleton, S. H., Srinivasan, A., & Sternberg, M. (1996). Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. Proceedings of the National Academy of Sciences, 93(1), 438–442.

King, R. D., Whelan, K. E., Jones, F. M., Reiser, P. G., Bryant, C. H., Muggleton, S. H., et al. (2004). Functional genomic hypothesis generation and experimentation by a robot scientist. Nature, 427(6971), 247–252.

Kipf, T.N., & Welling, M. (2017). 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

Kramer, S., Lavrač, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In Relational data mining, pp. 262–291. Springer

Krogel, M.,A., Rawles, S., Železnỳ, F., Flach, P. A., Lavrač, N., & Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In International Conference on Inductive Logic Programming. pp. 197–214. Springer

Kursuncu, U., Gaur, M., & Sheth, A. (2019). Knowledge infused learning (k-il): Towards deep incorporation of knowledge in deep learning. arXiv preprint arXiv:1912.00512

Lavrač, N., Džeroski, S., & Grobelnik, M. (1991). Learning nonrecursive definitions of relations with linus. In European Working Session on Learning. pp. 265–281. Springer

Lee, J., Lee, I., & Kang, J. (2019). Self-attention graph pooling. In International Conference on Machine Learning. pp. 3734–3743

Lodhi, H. (2013). Deep relational machines. In International Conference on Neural Information Processing. pp. 212–219. Springer

Marx, K. A., O'Neil, P., Hoffman, P., & Ujwal, M. (2003). Data mining the nci cancer cell line compound gi50 values: identifying quinone subtypes effective against melanoma and leukemia cell classes. Journal of chemical information and computer sciences, 43(5), 1652–1667.

McNaught, A. D., Wilkinson, A., et al. (1997). Compendium of chemical terminology (Vol. 1669). Oxford: Blackwell Science Oxford.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. Proceedings of the AAAI Conference on Artificial Intelligence., 33, 4602–4609.

Muggleton, S. (1995). Inverse entailment and progol. New generation computing, 13(3–4), 245–286.

Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., & Srinivasan, A. (2012). Ilp turns 20. Machine learning, 86(1), 3–23.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., & Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems. pp. 8024–8035

Plotkin, G. (1971). Automatic Methods of Inductive Inference. Ph.D. thesis, Edinburgh University

Prechelt, L. (1998). Early stopping-but when? In Neural Networks: Tricks of the trade, pp. 55–69. Springer

Ralaivola, L., Swamidass, S. J., Saigo, H., & Baldi, P. (2005). Graph kernels for chemical informatics. Neural networks, 18(8), 1093–1110.

Ramakrishnan, G., Joshi, S., Balakrishnan, S., & Srinivasan, A. (2007) Using ilp to construct features for information extraction from semi-structured text. In International Conference on Inductive Logic Programming. pp. 211–224. Springer

Saha, A., Srinivasan, A., & Ramakrishnan, G. (2012). What kinds of relational features are useful for statistical learning? In International Conference on Inductive Logic Programming. pp. 209–224. Springer

Sato, R. (2020). A survey on the expressive power of graph neural networks. arXiv preprint arXiv:2003.04078

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. IEEE Transactions on Neural Networks, 20(1), 61–80.

Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks, 8(3), 714–735.

Srinivasan, A. (2001). The aleph manual. https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html

Srinivasan, A., & King, R. D. (1999). Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. Data Mining and Knowledge Discovery, 3(1), 37–57.

Stevens, R., Taylor, V., Nichols, J., Maccabe, A. B., Yelick, K., & Brown, D. (2020). Ai for science. Tech. rep., Argonne National Lab.(ANL), Argonne, IL (United States).

Van Craenenbroeck, E., Vandecasteele, H., & Dehaspe, L. (2002). Dmax's functional group and ring library. https://dtai.cs.kuleuven.be/software/dmax/

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In International Conference on Learning Representations, https://openreview.net/forum?id=rJXMpikCZ

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? In International Conference on Learning Representations, https://openreview.net/forum?id=ryGs6iA5Km

Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., & Talukdar, P. (2019). Hypergcn: A new method for training graph convolutional networks on hypergraphs. In Advances in Neural Information Processing Systems. pp. 1509–1520

Zhang, D., Yin, J., Zhu, X., & Zhang, C. (2018). Network representation learning: A survey. IEEE transactions on Big Data

Zhou, D., Huang, J., & Schölkopf, B. (2007). Learning with hypergraphs: Clustering, classification, and embedding. In Advances in neural information processing systems. pp. 1601–1608

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C. & Sun, M. (2018). Graph neural networks: A review of methods and applications. arXiv preprint arXiv:1812.08434

Ziegler, K., Caelen, O., Garchery, M., Granitzer, M., He-Guelton, L., Jurgovsky, J., Portier, P.E., & Zwicklbauer, S. (2017). Injecting semantic background knowledge into neural networks using graph embeddings. In 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). pp. 200–205. IEEE

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.