Learning and Reasoning with Graph Data: Neural and Statistical-Relational Approaches

Manfred Jaeger
Aalborg University, Denmark

Abstract

Graph neural networks (GNNs) have emerged in recent years as a very powerful and popular modeling tool for graph and network data. Though much of the work on GNNs has focused on graphs with a single edge relation, they have also been adapted to multi-relational graphs, including knowledge graphs. In such multi-relational domains, the objectives and possible applications of GNNs become quite similar to what for many years has been investigated and developed in the field of statistical relational learning (SRL). This article first gives a brief overview of the main features of GNN and SRL approaches to learning and reasoning with graph data. It analyzes then in more detail their commonalities and differences with respect to semantics, representation, parameterization, interpretability, and flexibility. A particular focus will be on relational Bayesian networks (RBNs) as the SRL framework that is most closely related to GNNs. We show how common GNN architectures can be directly encoded as RBNs, thus enabling the direct integration of "low level" neural model components with the "high level" symbolic representation and flexible inference capabilities of SRL.

2012 ACM Subject Classification Computing methodologies → Logical and relational learning; Computing methodologies → Neural networks

Keywords and phrases Graph neural networks, Statistical relational learning

Digital Object Identifier 10.4230/OASIcs.AIB.2022.5

Category Invited Paper

1 Introduction

Learning and reasoning with graph and network data has developed as an area of increasing importance over recent years. Social networks, knowledge graphs, sensor and traffic networks are only some of the examples where graph structured data arises in important applications. Much of the attention currently focuses on *graph neural networks (GNNs)* as the technology for solving the challenges posed by this kind of data. While often very powerful in terms of scalability and predictive performance, graph neural networks suffer from the same drawbacks as other deep learning methods: lack of interpretability, limited support for the integration of prior domain knowledge, lack of robustness, and the inability to support more flexible reasoning than performing a specific task of prediction or synthetic graph generation. The field of *statistical relational learning (SRL)* has been concerned with learning and reasoning with graph and network data for over 20 years. Here the use of logic-based, symbolic representations and inference techniques, probabilistic graphical models, and relational database technology supports the construction of interpretable models via a combination of expert knowledge and machine learning, as well as a wide range of inference tasks, such as prediction and (most probable) explanations for varying and incomplete amounts of input data. On the other hand, SRL techniques lag behind GNNs in terms of scalability and predictive power in scenarios where the availability of extensive training data enables the training of the highly parameterized GNN models.

Combining the respective strengths of GNN and SRL technology is an emergent research area [49, 59, 69, 15]. Some works emphasize the complementarity of SRL and GNN approaches, and propose techniques that combine them in order to leverage the strengths of both [49, 59].

Table 1 Overview of notation.

$G = (V, E)$	Graph with vertices (a.k.a. domain) V and edges E
n	Cardinality of V
N_i	Neighbors of node $i \in V$
$\mathcal{A}$	signature of node attributes
$\boldsymbol{A}$	$n \times \lvert\mathcal{A}\rvert$ matrix (table) of attribute values
$\mathcal{R}$	signature of relations of varying arities
$\mathcal{G}(V, \mathcal{R})$	Set of all graphs for signature $\mathcal{R}$ over domain V
$\boldsymbol{h}^k(i)$	Representation vector at level (layer) k for node $i \in V$
n_k	Dimension of vectors $\boldsymbol{h}^k(i)$
m	Number of hidden layers
$\{\!\!\{ \ldots \}\!\!\}$	Delimiters for multisets
P_V	Distribution over all graphs (for a given signature $\mathcal{R}$) with domain (vertex set) V
$F_{r(X_1,\ldots,X_k)}$	Probability formula defining the (conditional) probabilities for the k-ary relation r

In contrast, [69] directly uses relational logic as a high-level and flexible specification tool for neural architectures with a generic underlying training technique, thus essentially showing how a logic-based framework subsumes (graph) neural technology. In this article we, too, shall emphasize the overlap rather than the complementarity of SRL and GNN technology. We first give a brief overview of key aspects of modeling graph data using SRL and GNN approaches. Looking more closely at their respective semantics we then obtain a basis for establishing a direct correspondence between GNNs and some types of SRL models (Section 6), which gives rise to a direct encoding of GNNs as *relational Bayesian networks* (Section 7 and 9). We illustrate by examples the benefits of such an embedding of GNNs in a general SRL framework with regard to

- neuro-symbolic integration: the ability to combine symbolically specified (expert) knowledge with numerical optimization in neural architectures.
- flexible reasoning capabilities: the support provided by SRL frameworks for probabilistic reasoning beyond solving a fixed prediction task.

We consider in greater detail selected central themes that have played an important role in GNN and SRL research:

- Expressivity: what are the capabilities and limitations for the discriminative power of GNN/SRL models (Section 8)?
- Homophily: how do different approaches model and exploit the typical homophily properties in (social) networks?
- Aggregation: what operations can be used to aggregate information provided by related entities in a graph (Section 11)?

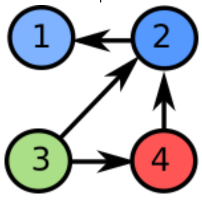
2 Graph Data

In its most basic form, graph data simply consists of a graph

$$G = (V, E) \tag{1}$$

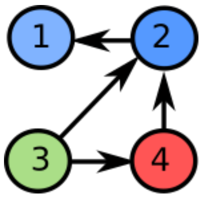
defined by a set of vertices V, and a set of (directed or undirected) edges E. We shall limit ourselves to graphs with finite V, and use $n = |V|$ to denote its cardinality. We can then also assume that $1, \ldots, n$ are unique identifiers for the vertices, and we simply use

Node	color
1	blue
2	blue
3	green
4	red

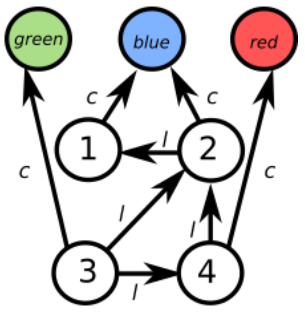


(a)

Node	c_blue	c_green	c_red
1	1	0	0
2	1	0	0
3	0	1	0
4	0	0	1



(b)



(c)

Figure 1 Three representations of node attributes: (a) categorical attribute table, (b) one-hot attribute table, (c) multi-relational representation with binary attribute relation $c(olor)$ distinguished from original $l(ink)$ relation.

$i, j, \ldots \in \{1, \ldots, n\}$ to refer to elements of V. In an *attributed graph* the nodes are labeled with a set of attributes, which can be either Boolean, categorical or numeric. In a social network whose nodes represent people, for example, node attributes could be *gender* and *age*. We can write a graph whose nodes are labeled with k different attributes as

$$G = (V, E, \boldsymbol{A}), \tag{2}$$

where $\boldsymbol{A}$ is an $n \times k$ attribute matrix whose ith row contains the attribute vector for vertex i. The term "matrix" here is used in a loose sense: when some of the attributes are categorical, then corresponding columns in the matrix have symbolic values, and $\boldsymbol{A}$ would somewhat more appropriately be referred to as an attribute "table". Using one-hot encodings of categorical attributes one can obtain a purely numerical attribute matrix $\boldsymbol{A} \in \mathbb{R}^{n \times k'}$. Figure 1 (a) and (b) illustrates these two alternative representations.

In many cases, graph models for real world networks require multiple edge relations. In a social network, for instance, there may be separate *friend* and *follower* connections between users. In a sensor or computer network, different types of connections (wireless, cable, ...) can be represented by different edge relations. This leads us to *multi-relational* graphs with several distinct edge relations. We write multi-relational graphs as

$$G = (V, \boldsymbol{E}, \boldsymbol{A}), \tag{3}$$

where now $\boldsymbol{E} = (E_1, \ldots, E_r)$ is a tuple of different edge relations. In the basic sense, edges are just either 'present' or 'absent', i.e., have a Boolean value. However, an edge labeling can also attach a categorical or numerical value to the edges, so that just like attributes, edge relations can be either Boolean, categorical, or numeric. Multi-relational graphs also provide an opportunity to represent node attributes as relations between the original graph vertices and attribute values that are materialized as additional nodes. This representation is illustrated in Figure 1 (c).

Going beyond binary edge relations, we can consider (directed) *hyperedges* $E \subseteq V^k$ for any $k \geq 3$. With few exceptions (e.g. [51]) hyperedges have not been considered in the GNN literature. From the predicate logic perspective of SRL, on the other hand, attributes, edges and hyperedges all are just relations with a certain *arity* $k \geq 1$. Under this uniform perspective of predicate logic, we can write an *attributed, multi-relational hypergraph* as

$$G = (V, \boldsymbol{R}) \tag{4}$$

where $\boldsymbol{R} = (R_1, \ldots, R_r)$ is a tuple of relations with arities $a_i \in \{0, 1, 2, 3, \ldots\}$ $(i = 1, \ldots, r)$, all of which can be of type Boolean, categorical, or numeric. We use relations of arity 0 as representations of global graph properties such as *connected*, or a label like *toxic* for a graph representing a chemical molecule.

Knowledge Graphs

Knowledge graphs are often presented as a set *KG* of *triplets* of the form (*source*, *relation*, *target*). Each such triplet describes a specific relationship between a source and a target entity. There are several ways to cast such a collection of triplets as a graph in one of the forms considered above. A first option is to consider each triplet as a hyperedge of a single relation of arity 3. For illustrative purposes, we may call this single relation the *fact* relation, and expand the triplet notation to explicitly write *fact*(*source*, *relation*, *target*). In this view, relations are also treated as nodes (entities). While this perspective is somewhat encouraged by the triplet notation for knowledge graphs, it is usually not the one that underlies graphical representations of knowledge graphs, where typically a triplet is illustrated by two nodes for the source and target entities, connected by an edge labeled with *relation*. The underlying graph model then is that of a multi-relational graph with binary relations, and the perhaps more appropriate notation for a triplet is the classical form of writing *relation*(*source*, *target*). Moreover, expressing everything as triplets also requires a representation of node attributes in the form illustrated in Figure 1 (c), i.e. in the form of relations *attribute*(*entity*, *attribute_value*). A specific feature of knowledge graphs compared to other multi-relational graphs is the number of distinct relations, which can easily lie in the thousands. Multi-relational graphs representing more specif domains (such as bibliographic or social networks), in contrast, mostly contain only a relatively small number of relations. When considering the application of machine learning solutions for graph data to knowledge graphs, one must therefore consider not only the ability of a solution to deal with multi-relational graphs, but also their scalability in terms of the number of relations.

3 Graph Neural Networks

We here give a high-level summary of some key features of graph neural networks. For more complete and details surveys also covering types of GNNs not considered here (e.g. graph auto-encoders), the reader is referred to [74, 79].

Graph neural networks compute *representations* of the nodes as real valued vectors. Also often referred to as *embeddings* or *feature vectors*, these representations can then be the basis for supervised tasks like node classification, graph classification and link prediction, or unsupervised tasks such as graph clustering. Often the representations are learned in an "end-to-end" fashion to support a particular task, but they can also be constructed in a stand-alone process in order to support a variety of downstream applications (e.g. the unsupervised version of GraphSAGE [24]).

In the following, we first assume that $G = (V, E, \boldsymbol{A})$ is an attributed graph as in equation (2). In a very general, abstract form, the computation of the node representations proceeds in multiple steps, starting with initial representations

$$\boldsymbol{h}^0(i) \in \mathbb{R}^{n_0} \qquad (i \in V). \tag{5}$$

The initial representations can be collected in an $n \times n_0$ matrix, denoted $\boldsymbol{H}^0$. Using $\boldsymbol{H}^0[i, \bullet]$ to denote the ith row of $\boldsymbol{H}^0$, we then have $\boldsymbol{h}^0(i) = \boldsymbol{H}^0[i, \bullet]$. This gives us a bit of notational redundancy, which can be quite convenient, however. We will consider the choice of initial representations in more detail below. For now it may be helpful to think of the attribute vector $\boldsymbol{h}^0(i) = \boldsymbol{A}[i, \bullet]$ as the initial representation.

Then, given a representation $\boldsymbol{H}^k$ computed in the kth step, the $(k+1)$th representation is obtained as a function

$$\boldsymbol{H}^{k+1} = F^k(\boldsymbol{H}^k, G) \tag{6}$$

of the previous representation and the graph structure G. In most GNN architectures, the computation of the representation $\boldsymbol{h}^{k+1}(i)$ for node i only depends on the rows of $\boldsymbol{H}^k$ corresponding to i itself, and its neighbors. Then (6) can be expressed as follows:

$$\boldsymbol{h}^{k+1}(i) = F^k(\boldsymbol{h}^k(i), \{\!\!\{\boldsymbol{h}^k(j) | j \in N_i\}\!\!\}, G), \tag{7}$$

where N_i denotes the set of graph neighbors of i, and the delimiters $\{\!\!\{\ldots\}\!\!\}$ indicate that this is a multiset. Based on this dependence of the representation update of a node i on its neighbors, the basic GNN computations have been described using *information diffusion* or *message passing* metaphors [67, 21]. A fundamental distinction arises according to whether the updates (7) are performed in a *recurrent* or *fixed* architecture. In the former, the update function F^k is the same for all k, and updates are performed until a fixed point $\boldsymbol{H}^{k+1} = \boldsymbol{H}^k$ is reached (e.g.[67]). In the latter, updates are performed for a fixed number of steps $k = 0, \ldots, m$, and each step may be defined by a different update function F^k. In this article we focus on fixed architectures, which also constitute the majority of current GNN frameworks. In any case, a final node representation $\boldsymbol{H}^m$ is used to compute an output function. In node classification applications, an output is computed for each node i, based on that node's representation:

$$\boldsymbol{o}(i) = O(\boldsymbol{h}^m(i)). \tag{8}$$

When the task is graph classification, then an output is computed based on the representations of all nodes in the graph. This particular type of function is usually referred to as a *readout*:

$$\boldsymbol{r} = R(\{\!\!\{\boldsymbol{h}(i) | i \in V\}\!\!\}). \tag{9}$$

Figure 2 gives a schematic illustration of the computation graph for a fixed architecture with $m = 2$ steps, both for a node classification and a graph classification scenario.

Link prediction tasks can be addressed by GNNs in several ways. The most direct one is to use a GNN architecture as shown in Figure 2, but without the final output or readout layers. Instead, link prediction is directly performed based on the representations of the nodes, for example simply using the dot product $\boldsymbol{h}^m(i) \cdot \boldsymbol{h}^m(j)$ as a predictive score for the existence of a link between i and j [68]. A very different approach is to turn the link prediction problem into a graph classification problem for the "enclosing subgraph" of the candidate edge [78].

Equation (7) is only an abstract description of the computation of $\boldsymbol{H}^{k+1}$. All concrete functions implementing this description need to deal with the multiset argument $\{\!\!\{\boldsymbol{h}^k(j) | j \in N_i\}\!\!\}$ which differs in cardinality for different nodes i. In order to turn this into a fixed dimensional input an *aggregation* function such as *sum* or *mean* is usually applied. Making this aggregation step explicit, we can re-write (7) as

$$\boldsymbol{h}^{k+1}(i) = F_1^k(\boldsymbol{h}^k(i), agg\{\!\!\{F_2^k(\boldsymbol{h}^k(j), G) | j \in N_i\}\!\!\}, G), \tag{10}$$

which also allows that the representations $\boldsymbol{h}^k(j)$ are transformed by a function F_2^k before aggregation.This transformation may also depend on the graph structure G. This, for example, enables a scaling of $\boldsymbol{h}^k(j)$ before aggregation by a factor $1/\sqrt{d_j}$ with d_j the degree

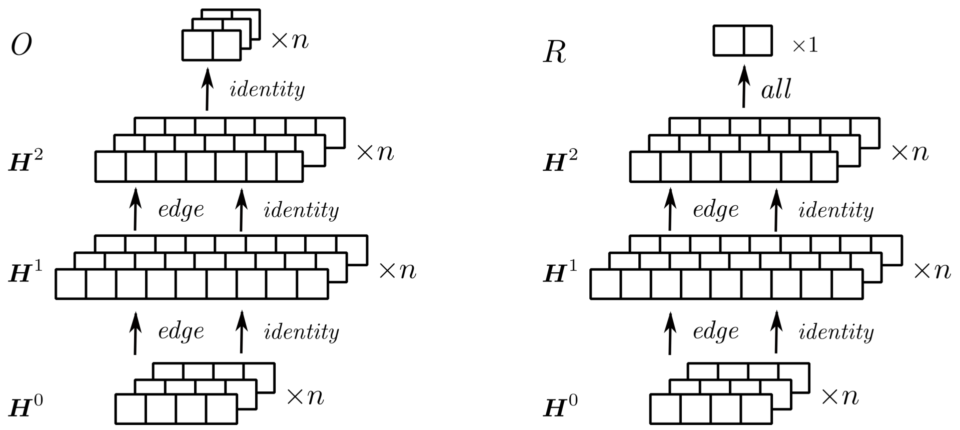


Figure 2 Computation graphs for basic fixed architecture showing the representation vectors $\boldsymbol{h}^k$ at each step, and indicating the relations *edge*, *identity* and/or *all* between nodes that defines the dependence of a representation at the next step on representations/output at the previous step. Left: architecture for node classification with one output per node; right: architecture for graph classification, with one global *readout* output function.

of j in G, which is required to cover the original graph convolutional network [41] within the formulation of (10). In most cases, the dependence of F_1^k, F_2^k on G will only be through the degrees of the nodes i, j, and often there is no such dependence at all.

The abstract functions F_1^k, F_2^k are defined in reality by neural network layers. These are usually quite simple in nature, and typically just consist of a linear function (layer) followed by a non-linear activation function.

Multi-Relational GNNs

The largest part of the GNN literature focuses on the case of attributed graphs with a single edge relation (2). Statistical relational learning, on the other hand, is concerned with rich structures as described by (4). In many applications, however, multi-relational graphs (3) are sufficient, and GNNs can quite easily be adapted to also deal with these [56, 68, 48]. The functional form (10) then becomes

$$\begin{aligned}\boldsymbol{h}^{k+1}(i) = F_1^k(\boldsymbol{h}^k(i), agg\{\!|F_{2,1}^k(\boldsymbol{h}^k(j), G)|j \in N_i^{E_1}|\!\}, \ldots,\\ agg\{\!|F_{2,r}^k(\boldsymbol{h}^k(j), G)|j \in N_i^{E_r}|\!\}, G), \quad (11)\end{aligned}$$

where now $N_i^{E_h}$ denotes the set of i's neighbors according to relation E_h.

We note here that we have limited our exposition to the most fundamental forms of GNN "message passing" architectures. This basic form has seen modifications and generalizations in many directions, including the addition of *attention mechanisms*[71], or *skip connections* that make computations at one layer dependent not only on the output of the previous layer. Most of the following considerations on the relationship between GNN and SRL models carry over to such more general forms of GNNs.

4 Statistical Relational Learning

Statistical relational learning (SRL) [20, 60] is a fairly diverse field encompassing different approaches to combine elements of relational logic representations, probabilistic graphical models, logic programming, and relational databases, for probabilistic reasoning and learning about entities and their relationships. SRL frameworks use the language of relational logic to represent basic facts in the form of *atomic expressions*, or *atoms* for short:

$$\mathit{friend}(\mathit{lars}, \mathit{giovanni}),\ \ \mathit{color\_green}(\mathit{emerald}),\ \ \mathit{connected}(X, \mathit{router\_48264}), \ldots$$

Generally, an atom is a relation symbol followed by a list of arguments with a length corresponding to the relation's *arity*. Arguments can be *constants* representing specific entities, or *variables*. A usual convention is that variables start with upper-case letters, whereas constants start with lower-case letters. Thus, in our examples above, X is a variable, and all other arguments are constants. An atom that only contains constants as arguments is called *ground*. It represents a specific fact that can either be true or false. In a pure predicate logic interpretation, atoms can only evaluate to Boolean values. Thus, the color of an entity would need to be represented in the form $\mathit{color\_green}(X)$ or $\mathit{color}(X, \mathit{green})$, as in Figure 1 (b) or (c). However, in SRL frameworks this can often be loosened to also allow categorical atoms such as $\mathit{color}(X) \in \{\mathit{red}, \mathit{green}, \mathit{blue}\}$, or numerical atoms such as $\mathit{length}(X) \in \mathbb{R}$. It is thus apparent that the fundamental building blocks of SRL frameworks describe multi-relational hypergraphs as in (4). However, depending on the SRL context and background, these kinds of structures also go by very different names, such as *Herbrand interpretations* [12], or *possible worlds* [62]. In order to maintain a close connection with the preceding sections, we shall here continue to speak about graphs (which always are understood to be multi-relational hypergraphs).

SRL frameworks define probability distributions over graphs. More specifically, consider a fixed set $\mathcal{R}$ of relations (of different arities). We also call $\mathcal{R}$ a *signature* of relation symbols. Let V be a finite set of vertices (more commonly referred to as a *domain* in SRL contexts), and let $\mathcal{G}(V, \mathcal{R})$ be the set of all graphs with vertex set V for the relations $\mathcal{R}$. An *SRL model* then defines a mapping that assigns to every finite set V a probability distribution over $\mathcal{G}(V, \mathcal{R})$ [34].

▶ **Example 1.** Classic random graph models such as the basic Erdős-Rényi model [17], the stochastic blockmodel [27], or the preferential attachment model [3], are SRL models in our sense with a signature $\mathcal{R}$ consisting of a single binary $\mathit{edge}(X, Y)$ relation (however, this does not mean that every SRL framework as detailed below is necessarily able to capture all of these random graph models). All these models define for every cardinality $|V| = n$ a probability distribution over all graphs with n nodes.

We call an *SRL framework* any specific system of representation and inference tools for SRL models. More specifically, an SRL framework provides:

- *Syntax and semantics*: a formal language for the specification of SRL models, and a semantic specification of the probability distribution that is defined by the model.
- *Inference*: general algorithms for computing for a given vertex set V, and two subsets $A, B \subseteq \mathcal{G}(V, \mathcal{R})$, the conditional probability $P_V(A|B)$ under the distribution P_V defined by the model for the domain V. Different frameworks will differ in how the subsets A, B can be defined, but all will usually allow to specify sets by ground atoms: $P_V(\mathit{republican(mary)}|\mathit{friends(mary,carl)})$ will then stand for $P_V(A|B)$, with A the set of all graphs where the (Boolean) attribute *republican* is true for entity *mary*, and B the

set of all graphs where the relation *friends* holds between the entities *mary* and *carl* (assuming that V contains entities *mary* and *carl*, and that the signature $\mathcal{R}$ contains the relations *republican* and *friends*).

- *Learning*: methods for statistical learning of an SRL model from data. One here distinguishes *parameter learning* and *structure learning.* The latter refers to learning the high-level, symbolic part of the model specification, and the former to fitting numeric parameters of the model.

Within the very diverse landscape of SRL frameworks, one can identify a number of major paradigms, which we briefly survey in the following (a slightly more detailed exposition along similar lines can be found in [32]). For the following we assume that all relations are Boolean or categorical, which is the original and main focus of SRL frameworks.

Bayesian Network Constructors

Here the actual distribution P_V for a given V is eventually represented by a Bayesian network whose nodes are all the ground atoms that can be formed from relations in $\mathcal{R}$ with entities from V. The SRL model provides a general blueprint for how such a Bayesian model representation is constructed for any domain V. Languages for defining such blueprints fall into two main categories: *rule based* and *graphical templates.* The basic building blocks of rule based approaches are logical implications between atoms, such as

$$\textit{infected}(X) \leftarrow \textit{contact}(X,Y) \tag{12}$$

which are further annotated with quantitative probabilistic information. The qualitative parts of the rules (as shown in (12)) then define the graphical dependency structure in the Bayesian network for P_V, whereas the probabilistic annotations (not shown) define the quantitative conditional probability specifications. As (12) illustrates, the rules will usually only contain variable symbols, not constants referring to specific entities, and thereby are applicable to arbitrary domains V. This brand of SRL frameworks has its roots in what originally was called *knowledge-based model construction* [7, 53], and is further represented by *Bayesian logic programs* [37] and *relational Bayesian networks* [29].

As illustrated by (12), the directed dependencies expressed by the rules often coincide with causal dependencies. However, it is not generally necessary that the rules have a causal background, or even that they comply with an existing causal direction (it would be perfectly valid, if rather counter-intuitive, to construct a model including a rule $\textit{infected}(X) \leftarrow \textit{fever}(X)$ that inverts the causal direction).

Template-based frameworks follow essentially the same modeling paradigm, but using graphical representations of Bayesian network fragments as the basic representational building block [46, 45]. Another paradigm for abstract graphical representations that can be compiled into Bayesian networks is based on entity-relationship diagrams as used in relational databases [18, 25].

Markov Network Constructors

This type of SRL frameworks is mostly represented by *Markov logic networks* [62], which are closely related to *exponential random graph models* that have a long history in statistics and discrete mathematics. Markov logic networks also use logic-based representations to specify blueprints for the construction of a probabilistic graphical model for a specific distribution P_V. However, the target model now is an undirected Markov network, rather than a directed Bayesian network. Instead of directed implications, the logical building blocks are disjunctions (a.k.a. clauses) of atoms (possibly negated):

$$\neg \mathit{friends}(X,Y) \lor \neg \mathit{republican}(X) \lor \mathit{republican}(Y), \tag{13}$$

which also are annotated with numerical weights. Such a clause represents a Boolean *feature* of entity pairs (X,Y), and the associated weight specifies whether graphs in which this feature holds for many concrete entity pairs are more or less probable. Instead of directed dependencies as in Bayesian networks, these features can now specify undirected, symmetric (non-causal) dependencies. The clause in (13), for example, can be rewritten as $\mathit{friends}(X,Y) \rightarrow (\mathit{republican}(X) \rightarrow \mathit{republican}(Y))$, and thus expresses a *homophily* features: friends are likely to have the same political leanings.

Probabilistic Logic Programming

Another major line of SRL developments is rooted in logic programming, and the machine learning tradition of *inductive logic programming.* A logic program such as

$$\begin{array}{l} \mathit{edge}(a,b) \\ \mathit{edge}(b,c) \\ \mathit{path}(X,Y) \leftarrow \mathit{edge}(X,Y) \\ \mathit{path}(X,Y) \leftarrow \mathit{edge}(X,Z), \mathit{path}(Z,Y) \end{array} \tag{14}$$

defines a unique *least Herbrand model*, which in our terminology is just a multi-relational graph. For the program (14) this is the graph over $V = \{a,b,c\}$ in which the relation *edge* contains the tuples $(a,b),(b,c)$, and the relation *path* contains the tuples $(a,b),(b,c),(a,c)$. In probabilistic logic programming the clauses are annotated with probabilities. Randomly selecting clauses according to their probabilities then induces a probability distribution over logic programs, and hence a probability distribution P_V over Herbrand models over V. As described here, (14) would only define a probability distribution P_V over the fixed domain $V = \{a,b,c\}$, and thus lack the generality we required of an SRL framework. However, concrete constants are usually only included in listings of simple ground facts, whereas general modeling rules are formulated at the generic level using only variables. This makes the framework still modular, and by substituting other sets of ground facts, the generic model can be applied to arbitrary domains V. Early examples of this probabilistic logic programming approach are [66, 58]. A more recent system at a very mature level of development is the ProbLog framework [39].

Inference

All the frameworks outlined above support to compute conditional probabilities of the form

$$P_V(q|e_1,\ldots,e_m), \tag{15}$$

where $q, e_1, \ldots, e_m$ all are ground atoms. There is no fundamental conceptual difference between graph classification, node classification, or link prediction, which are only distinguished by the arity of the query atom q.

▶ **Example 2.** Let V be a domain of n individuals. For some individuals we have made observations on the *infected* attribute, as well as on *contact* relations. For an individual *mary* we want to predict whether she is infected. This would be accomplished by computing the node classification query

$$P_V(\mathit{infected}(\mathit{mary})|\mathit{contact}(\mathit{mary},\mathit{john}), \mathit{contact}(\mathit{john},\mathit{anne}), \mathit{infected}(\mathit{john}), \neg\mathit{infected}(\mathit{anne}))$$

In this example the input information for the query atom $\mathit{infected}(\mathit{mary})$ only consists of the observations of four ground atoms related to entities in V. However, it may very well be the case that much more about the graph is known. For example, the *contact* relation might be fully observed, in which case the query would be conditioned on the whole *contact* graph.

Suppose, conversely, that we have comprehensive observations of the *infected* attribute, and want to infer the *contact* relations. This would lead to link prediction queries such as

$$P_V(\mathit{contact}(\mathit{john}, \mathit{anne}) | \mathit{infected}(\mathit{mary}), \neg\mathit{infected}(\mathit{john}), \neg\mathit{infected}(\mathit{anne})).$$

Note that the model used for answering this query is the same generative model P_V as in the node classification query before. Finally, suppose we have a predicate *spreading* that represents for the whole population V whether the infection is currently spreading in V. A probabilistic model for *spreading* might be based on the number of pairs of individuals in V that are in contact, and one of which is infected. Then a query like

$$P_V(\mathit{spreading}() | \mathit{infected}(\mathit{mary}), \neg\mathit{infected}(\mathit{john}), \mathit{contact}(\mathit{john}, \mathit{anne}), \mathit{contact}(\mathit{mary}, \mathit{john}))$$

would predict the *spreading* status of the whole domain (this query in a realistic scenario would be conditioned on much more comprehensive input observations than shown here). To emphasize the view of the query predicate as a relation of arity 0, we write it here in the format $\mathit{spreading}()$.

SRL frameworks in the Bayesian or Markov network constructor classes can use existing inference algorithms for these types of probabilistic graphical models for the computation of arbitrary queries (15). However, constructing the graphical model for the full distribution P_V in order to compute the query probability (15), which often only refers to a small subset of the entities in V, may be very wasteful. Significant effort, therefore, has been spent on the question whether inference could be performed more directly on the basis of the high-level SRL model specification, instead of its compilation into a standard graphical model [30, 57, 13]. However, decisive breakthroughs in this area of so-called *lifted inference* remain elusive.

Inference for probabilistic logic programming frameworks is somewhat different in nature. Here the probability of the query atom q will depend on which of the possible randomly sampled programs support the derivation of the query atom q. The calculation of the query probability therefore is essentially based on the construction of all possible proofs of q from the rules and facts in the probabilistic logic program, and then a calculation of the probability that a given proof is actually supported by a randomly sampled subset of the facts and rules.

Learning

Learning of SRL models can be separated into learning the *structure* of the high-level symbolic component of the model, and learning numerical *parameters*. For the parameter learning task often quite effective methods exist. In many cases, also learning from incomplete data is supported by statistical learning techniques like expectation-maximization. Structure learning, on the other hand, amounts to a search in a complex combinatorial space of symbolic representations. Full structure learning without any prior constraints provided by (expert) domain knowledge is certainly not a solved problem at this point. We will come back to this issue in Section 12.

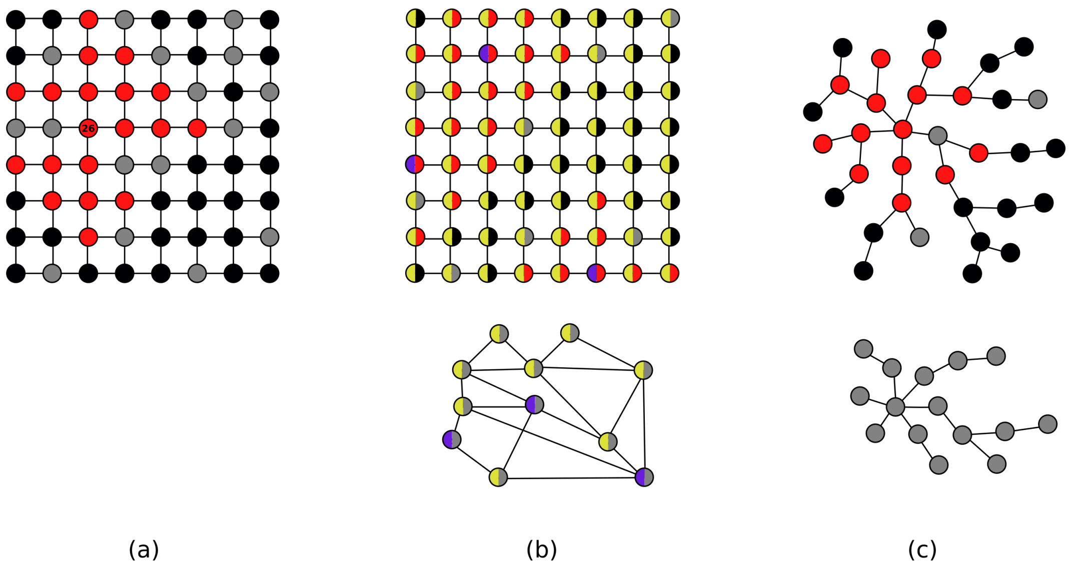


Figure 3 Transductive and inductive scenarios. (a): transductive, learnable from node identifiers; (b): inductive, learnable from node attributes; (c): inductive, learnable from graph structure. Graphs for training and (transductive) prediction in the top row; new test graphs underneath.

5 Transductive and Inductive Inference

Before we investigate at greater depth the relationship between GNN and SRL models we briefly discuss the difference between *transductive* and *inductive* inference settings. Roughly speaking, the former describes scenarios where at the time of model learning the nodes for which predictions are going to be made are already known. The latter describes scenarios where a predictive model is learned from training data, and that predictive model later is applied to formerly unknown nodes, or completely new graphs. Figure 3 illustrates this difference. In all graphs the target of prediction is a class label with values 'red' and 'black'. Figure 3 (a) shows a partially labeled graph where nodes do not have attributes other than the class label. Unlabeled nodes are shown in gray. For the particular unlabeled nodes in this graph one could learn to predict that a node is red if it is at most 3 hops away from node '26'. This model makes its predictions based on the relationship to a specific node in the training graph, and therefore does not generalize to other graphs. The learning scenario and the constructed model hence are transductive. The top of Figure 3 (b) shows a partially labeled graph where nodes also have an attribute *color* with values 'yellow' and 'blue'. This attribute here is assumed to be observed for all nodes. One could here learn a model that predicts a node to be 'red' if it is at most 2 hops away from a blue node. This model can be used to both classify the unlabeled nodes in the training graph shown in the top of the figure, as well as the nodes in a completely new and unlabeled graph shown underneath. The ability for inductive generalization is not always dependent on the existence of node attributes. Figure 3 (c) (top) shows a partially labeled graph without any node attributes. In this case one could learn to predict a node to be red if it is at most 2 hops away from a node that has a degree ≥ 5. This model would again be able to classify nodes in completely new and unlabeled graphs, like the one shown at the bottom or Figure 3 (c).

SRL models as we have described them in Section 4 are inductive in nature: they define for arbitrary new node sets V a probability distribution (and hence a prediction model) over $\mathcal{G}(V, \mathcal{R})$. This implies that SRL models cannot be defined in terms of particular node identifiers such as “node 26” in Figure 3 (a), which is reflected in syntax rules according to which elements like (12) or (13) can not contain constants denoting specific domain entities (and recall that in a slightly modified form, this constraint also applies to probabilistic logic programming models (14)). This focus on inductive models is more a historic convention than a necessary feature of SRL models: for example, a version of a rule like (12)

$$\mathit{blue}(X) \leftarrow \mathit{edge}(X, Y_1) \wedge \mathit{edge}(Y_1, Y_2) \wedge \mathit{edge}(Y_2, \mathit{node}_{26})$$

that contains references to a specific node would be able to express our transductive model for Figure 3 (a).

Whether a GNN model is inductive or transductive essentially depends on the initial representations $\boldsymbol{h}^0(i)$ used as inputs to the model. If these initial representations are node attribute vectors (as tacitly assumed in Section 3) then the resulting model will be inductive and able to handle scenarios like the one in Figure 3 (b). However, GNNs can also operate on initial node representations that are one-hot-encodings of node identifiers, which then leads to transductive models suitable to handle the situation in Figure 3 (a). In scenarios as depicted in Figure 3 (c), neither node identifiers nor node attributes would be available as initial representations. In this case one an think of $\boldsymbol{h}^0(i)$ as consisting of a constant not depending on i. The representation $\boldsymbol{h}^1(i)$ obtained after one round of aggregation would then already be able to encode the degree of the node i, which with two additional layers then enables our predictive model for Figure 3 (c).

In the following detailed comparison of SRL and GNN models we focus on inductive versions for both types of modeling frameworks, noting that the analogies we obtain in the inductive setting will carry over to transductive scenarios when for SRL frameworks we permit the use of node identifiers.

6 Semantics: a comparison

For SRL frameworks we have identified a common, well-defined semantics. Denoting the set of probability distributions over $\mathcal{G}(V, \mathcal{R})$ as $\Delta\mathcal{G}(V, \mathcal{R})$, we can write this semantics as a mapping

$$V \mapsto \Delta\mathcal{G}(V, \mathcal{R}). \tag{16}$$

▶ **Example 3.** Consider a relational signature $\mathcal{R} = \mathit{follower}, \mathit{influencer}$, where *follower* is a Boolean binary relation, and *influencer* is a Boolean node attribute. Let V be any fixed, finite domain. A distribution $P_V \in \Delta\mathcal{G}(V, \mathcal{R})$ can be defined by first defining the distribution over the *follower* relation, and then the conditional distribution of the *influencer* attribute given the *follower* relation, i.e., factorizing the joint distribution over both relations as

$$P_V(\mathit{follower}, \mathit{influencer}) = P_V(\mathit{follower}) \cdot P_V(\mathit{influencer}|\mathit{follower}). \tag{17}$$

The simplest way to define $P_V(\mathit{follower})$ is via an Erdős-Rényi random graph model, according to which each edge $\mathit{follower}(i, j)$ has a constant probability $p \in [0, 1]$ of being true, regardless of the cardinality of V:

$$P_V(\mathit{follower}(i, j)) = p. \tag{18}$$

Observe here the subtle difference: $P_V(\mathit{follower}(i, j))$ in (18) denotes the probability for the single ground atom (i.e., Boolean random variable) $\mathit{follower}(i, j)$, whereas $P_V(\mathit{follower})$ in (17) denotes the distribution over the whole *follower* relation, i.e., the joint distribution over all $\mathit{follower}(i, j)$ atoms with $i, j \in V$.

Assuming that *follower*(i,j) means that i is a follower of j, then the probability of *influencer*(i) given the *follower* relation could be defined as a function of i's in-degree (number of incoming edges) in this relation, which we denote as $d_i^{in,follower}$. For example, one could define

$$P_V(\mathit{influencer}(i)|\mathit{follower}) = 1 - q^{d_i^{in,follower}}, \tag{19}$$

for some $q \in [0,1]$. Again note that the left side of (19) denotes the conditional probability of the ground atom *influencer*(i) given the complete specification of the *follower* relation over the whole domain. For the specification on the right side of (19) we extract from the *follower* relation the in-degrees of nodes as relevant features. (19) is a standard *noisy-or* model for independent causal influences. The combined model composed of (18) and (19) would be supported by almost every SRL framework in the Bayesian network constructor family. Also a GNN implementing (18) as a link prediction model would be easy to construct. (19) is based on an underlying probabilistic interpretation, which leads to aggregation by multiplication, rather than summation. This would typically not be supported by standard GNN architectures. However, a slightly different function such as

$$P_V(\mathit{influencer}(i)|\mathit{follower}) = \sigma(d_i^{in,follower}), \tag{20}$$

where σ denotes the sigmoid function, can be easily implemented as a node classification GNN.

The preceding example has illustrated how the definition of a generative SRL model can be accomplished by the specification of several conditional probability distributions, each of which resembles a node classification or link prediction model, similar to what can be implemented in the form of a GNN. In the following we analyze this relationship more closely.

GNNs tend to be defined in terms of their computational architecture, rather than a semantic specification of the types of functions they compute. For our purpose, however, it is important to clarify the mathematical structure of GNN functions. The nature of the output computed by a GNN depends on their use for a node classification, link prediction, or graph classification application. In all cases, however, a node-level representation is the crucial (intermediate) output of a GNN (i.e. the representations $\boldsymbol{H}^2$ in Figure 2). More critical than the question of what outputs a GNN computes is the question on what class of inputs it can operate. For now we shall limit our considerations to the case of GNNs operating on simple attributed graphs $G = (V, E, \boldsymbol{A})$. Let $\mathcal{A}$ denote a fixed set of attributes. For a given set V we then denote by $\mathcal{G}(V, \mathcal{E}, \mathcal{A})$ the set of all graphs with node set V, and attributes $\mathcal{A}$.

Figure 2 may suggest that the inputs to a GNN are feature vectors $\boldsymbol{H}^0[i, \bullet]$ for all nodes i. The GNN would then accept as inputs all graphs $(V, E, \boldsymbol{A})$ with fixed V, E, but arbitrary node attribute values $\boldsymbol{A}$. This is also consistent with the functional representations (7),(8), respectively (7),(9), if one interprets the occurrence of G here as fixed parameters of the function, not as inputs that can be varied freely. To obtain a clearer picture, we have to leave the high level of abstraction adopted in Section 3, and consider the specific functions usually implementing the function pattern (7) (or the already slightly more specific version (10)). The most basic form of this function is

$$\boldsymbol{h}^{k+1}(i) = f\left(\boldsymbol{W}^k\boldsymbol{h}^k(i) + \boldsymbol{U}^k \sum_{j \in N_i} \boldsymbol{h}^k(j) + \boldsymbol{b}^k\right) \tag{21}$$

where $\boldsymbol{W}^k, \boldsymbol{U}^k$ are $(n_{k+1} \times n_k)$-dimensional matrices, $\boldsymbol{b}^k$ is a n_{k+1}-dimensional (bias) vector (where n_k denotes the dimension of the node representations at step k), and f is an element-wise activation function (cf. e.g. [23, Equation (5.7)], [4, Equation (2)]).

In the multi-relational version (11) this becomes

$$\boldsymbol{h}^{k+1}(i) = f\left(\boldsymbol{W}^k\boldsymbol{h}^k(i) + \sum_{h=1}^{m}\boldsymbol{U}_h^k \sum_{j\in N_i^{E_h}} \boldsymbol{h}^k(j) + \boldsymbol{b}^k\right), \tag{22}$$

which captures the essence of e.g. [56, Equation 4] or [68, Equation 2] (but omitting normalization operations in the aggregations).

The learnable model parameters here are the $\boldsymbol{W}^k, \boldsymbol{U}^k, \boldsymbol{b}^k$. If these matrices/vectors have dimensions that do not depend on $n = |V|$, a trained model can be applied to graphs $(V, E, \boldsymbol{A})$ with arbitrary V and E. A case where these matrices may actually be specific for a particular $|V|$ is when the initial representation $\boldsymbol{h}^0$ is a one-hot-encoding of node identifiers. This, however, only makes sense in a transductive learning setting. In inductive scenarios, the model must be able to generalize to graphs of sizes other than the size of the training graph, and the G arguments in (7),(10) must be seen as free inputs of the GNN model. Assuming now such an inductive setting, and focusing on the node embedding functionality of GNNs, one can describe the semantics of GNNs that can take (multi-relational) graphs with arbitrary V as inputs as a mapping

$$\bigcup_V \mathcal{G}(V,\mathcal{R}) \rightarrow \bigcup_{n\geq 1} \mathbb{R}^{n\times n_m}, \tag{23}$$

such that an attributed graph $(V, E, \boldsymbol{A})$ or multi-relational graph $(V, \boldsymbol{R})$ is mapped to an output in $\mathbb{R}^{|V|\times n_m}$ (n_m being the dimension of the final node embedding vectors). If one rather considers the end-to-end semantics of a GNN as a node classifier, link predictor, or graph classifier, then the semantics becomes

$$\bigcup_V \mathcal{G}(V,\mathcal{R}) \rightarrow \bigcup_{n\geq 1} \mathbb{R}^{n^c\times k}, \tag{24}$$

where $c = 1$ for node classification, $c = 2$ for link prediction, $c = 0$ for graph classification, and k is the number of possible (node/edge/graph) classes.

Comparing SRL semantics (16) and inductive GNN semantics (23) or (24), we already observe that both are functions defined on (finite) multi-relational graphs for a given signature, i.e. the space $\bigcup_V \mathcal{G}(V,\mathcal{R})$. However, an SRL model computes for an input graph $(V, \boldsymbol{R})$ a probability $P_V(\boldsymbol{R})$, whereas on the right-hand side of (24) we find a matrix of feature vectors for nodes, edges, or the whole graph. To bring these two types of function values together, we take a closer look at how probability distributions on $\mathcal{G}(V,\mathcal{R})$ can be defined following the factorization strategy already illustrated in Example 3.

Given a fixed V, a multi-relational relational graph $G = (V, \boldsymbol{R})$ is determined by the definitions of the relations $\boldsymbol{R} = R_1, \ldots, R_r$ over V. Let P_V denote the probability distribution over $\Delta\mathcal{G}(V,\mathcal{R})$ defined by an SRL model. Then P_V can be factored according to the chain rule [44, Sec. 2.1.2.2] as

$$P_V(\boldsymbol{R}) = P_V(R_1)\cdot P_V(R_2|R_1)\cdot\ldots\cdot P_V(R_h|R_1,\ldots,R_{h-1})\cdot\ldots\cdot P_V(R_r|R_1,\ldots,R_{r-1}). \tag{25}$$

Equation (25) is a probabilistic law that holds for any distribution $P_V \in \Delta\mathcal{G}(V,\mathcal{R})$, no matter how P_V is defined or learned. In the following we write $R_{1:k}$ for the tuple of relations $R_1, \ldots, R_k$, and $\mathcal{R}_{1:k}$ for the corresponding signature of relation symbols. For the class of SRL frameworks that we have described as 'Bayesian network constructors', the chain rule also serves as the core of the representation strategy: in these SRL frameworks, the specification of the distribution P_V is decomposed into specifications of conditional probability distributions. This decomposition need not be performed at the level of whole relations as in (25). Often one rather factors the distribution at the level of the ground atoms: a definition of relation R_h over V is equivalent to a truth assignments to all ground atoms $R_h(\boldsymbol{i}) \mapsto \{\mathit{true}, \mathit{false}\}$ ($\boldsymbol{i} \in V^{\mathit{arity}(R_h)}$; for the case of R_h being a Boolean relation). Furthermore, also the definition of a relation-level factor $P_V(R_h|R_{1:h-1})$ can most easily be done at the level of ground atoms in the form

$$P_V(R_h|R_{1:h-1}) := \prod_{\boldsymbol{i} \in V^{\mathit{arity}(R_h)}} P_V(R_h(\boldsymbol{i})|R_{1:h-1}). \tag{26}$$

Note that (26) now is based on the assumption that ground atoms in the relation R_h are conditionally independent given the relations $R_{1:h-1}$. In the form (26) the specification of $P_V(R_h|R_{1:h-1})$ becomes a mapping of the form

$$\mathcal{G}(V, \mathcal{R}_{1:h-1}) \to [0,1]^{n^{\mathit{arity}(R_h)}}. \tag{27}$$

This is still assuming that R_h is Boolean, and thus $P_V(R_h(\boldsymbol{i})|R_{1:h-1})$ is given by a single probability value for $R_h(\boldsymbol{i})$ being *true*. With a little additional notation, this generalizes to arbitrary categorical R_h. The factorization strategy employed by an SRL model for P_V, and the specification of $P_V(R_h|R_{1:h-1})$ will be uniform across different V. We can therefore also say that an SRL framework in the Bayesian network constructor class defines for each relation R_h a mapping

$$\bigcup_V \mathcal{G}(V, \mathcal{R}_{1:h-1}) \to \bigcup_{n \geq 1} [0,1]^{n^{\mathit{arity}(R_h)}}, \tag{28}$$

which now is almost identical to (24). It becomes completely identical, if we assume that the GNN uses softmax normalization on its output to also generate a probability distribution over node or link labels.

To summarize: the specification of a generative distribution P_V can be accomplished via (25) as a series of probabilistic node classification and link prediction operations (and possibly predictions of relations of arity higher than 2, if such are present in $\mathcal{R}$). If one further assumes that predictions for the atoms of each relation are independent of each other, then an SRL model that uses the chain rule as its underlying representation paradigm essentially consists of r functions, each of which is equivalent to a GNN function.

In this section we have focused on a comparison of the SRL generative models with (discriminative) GNN models that are designed for specific prediction tasks. GNN models have also been proposed for graph generation [42, 76, 47, 11], and one may wonder whether these are not a more natural point of reference for comparison with SRL frameworks. This is not the case, however: most of the proposed generative GNN models are in the tradition of classic random graph models, and their primary purpose is to learn GNN models from which random graphs can be effectively sampled, such that the distribution of key graph statistics (e.g., degree distribution, clustering coefficients) in the randomly sampled graphs matches the distribution in the training data. This is different from the typical SRL scenario, where the purpose is not to generate full graphs according the distribution P_V, but to answer queries of the form (15). Thus, the objectives for which generative SRL models are built are much more aligned with the objectives of predictive GNN models, than with the objectives of generative GNN models. An exception to this observation is [42] where the generative model is applied to link prediction. This, however, is in a transductive setting where the generative model is only fitted to a single graph for which then missing links are predicted, thus again being very different in nature from SRL.

7 RBNs

We now review the SRL framework of *relational Bayesian networks (RBNs)* [29, 34], which directly follows the representation paradigm outlined by (25) and (26). The RBN language consists of a syntax for logic-functional expressions called *probability formulas* for the specification of the conditional probabilities $P_V(R_h(\boldsymbol{i})|R_1,\ldots,R_{h-1})$ appearing in (26). We here give a description of the RBN language that uses a slightly more verbose syntax than the one introduced in the original papers. The difference is entirely cosmetic, however, and consists of little more than an alternative choice of terminal symbols in the grammar. The language of probability formulas consists of four different syntactic constructs. For each construct we define the syntax, and the semantics that defines how for a tuple $\boldsymbol{i} \in V^{arity(R_h)}$ in a graph $G = (V, R_1, \ldots, R_{h-1})$ the formula evaluates to the probability $P_V(R_h(\boldsymbol{i})|R_1,\ldots,R_{h-1})$. In the following, F denotes a probability formula, and $\mathit{eval}(F, \boldsymbol{i}, G)$ the probability value it defines for $\boldsymbol{i}$ in G.

Constants

For a real number $q \in [0, 1]$

$$F \equiv q \tag{29}$$

is a probability formula. $\mathit{eval}(F, \boldsymbol{i}, G) = q$ for all $\boldsymbol{i}, G$.

Atoms

For a relation R_j with $j \in 1, \ldots, h-1$, and variable symbols $Y_1, \ldots, Y_{arity(R_j)}$

$$F \equiv R_j(Y_1, \ldots, Y_{arity(R_j)}) \tag{30}$$

is a probability formula with the semantics

$$\mathit{eval}(F, \boldsymbol{i}, G) = \begin{cases} 1 & \text{if } R_j(\boldsymbol{i}) \text{ is true in } G \\ 0 & \text{if } R_j(\boldsymbol{i}) \text{ is false in } G \end{cases} \tag{31}$$

This innocuous definition is quite significant, as it transforms logic-symbolic information represented by a relation R_j into numeric data.

WIF-THEN-ELSE

Assume that F_1, F_2, F_3 are probability formulas. Then

$$F \equiv \texttt{WIF}\ \ F_1\ \ \texttt{THEN}\ \ F_2\ \ \texttt{ELSE}\ \ F_3 \tag{32}$$

is a probability formula. 'WIF' here stands for "weighted if". The semantics is a weighted mixture of probabilities:

$$\begin{aligned} &\mathit{eval}(\texttt{WIF}\ \ F_1\ \ \texttt{THEN}\ \ F_2\ \ \texttt{ELSE}\ \ F_3, \boldsymbol{i}, G) = \\ &\qquad\qquad \mathit{eval}(F_1, \boldsymbol{i}, G)\mathit{eval}(F_2, \boldsymbol{i}, G) + (1 - \mathit{eval}(F_1, \boldsymbol{i}, G))\mathit{eval}(F_3, \boldsymbol{i}, G). \end{aligned} \tag{33}$$

Before we move on to the fourth and most important construct for probability formulas, we illustrate the use of the ones introduced so far.

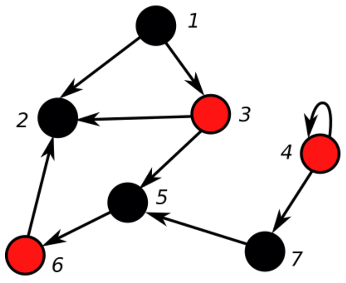


Figure 4 A small graph for two relations.

▶ **Example 4.** Figure 4 shows a small graph G for two relations consisting of a node attribute $R_1 = \mathit{red}$, and a binary relation $R_2 = \mathit{edge}$. Here the color black is just the negation of the Boolean attribute *red*. As always in SRL frameworks, relations are defined on ordered tuples, that means edges are directed.

Let now *positive* be a new Boolean node attribute. Associating with *positive* the constant probability formula $F_{positive(X)} \equiv 0.3$ would define a probability distribution according to which each node i has a constant probability of 0.3 of being *positive*. If *new_edge* is a new binary relation, then, similarly, $F_{new\_edge(X,Y)} \equiv 0.3$ would define a distribution over the *new_edge* relation according to which $\mathit{new\_edge}(X, Y)$ is true with probability 0.3 for each pair $i, j \in \{1, \ldots, 7\}$.

For a slightly more interesting example, let

$$F_{positive(X)} \equiv \texttt{WIF } \mathit{red}(X) \texttt{ THEN } 0.3 \texttt{ ELSE } 0.9.$$

Then $\mathit{eval}(F_{positive(X)}, i, G) = 0.3$ for the red nodes $i = 3, 4, 6$ of G, and $\mathit{eval}(F_{positive(X)}, i, G) = 0.9$ for the non-red nodes. For *new_edge* we could also define

$$F_{new\_edge(X,Y)} \equiv \texttt{ WIF } \mathit{edge}(Y, X) \texttt{ THEN } 0.5 \texttt{ ELSE } 0.$$

This specification would add for every edge (j, i) in the existing *edge* relation with probability 0.5 the reverse edge (i, j) to the *new_edge* relation. We can add the condition that in the existing edge the source node must be red in order for the reverse edge to be generated:

$$\begin{aligned} F_{new\_edge(X,Y)} \equiv \quad & \texttt{WIF } \mathit{edge}(Y, X) \texttt{ THEN WIF } \mathit{red}(Y) \\ & \qquad\qquad\qquad\qquad \texttt{THEN } 0.5 \\ & \qquad\qquad\qquad\qquad \texttt{ELSE } 0 \\ & \qquad\qquad\quad \texttt{ELSE } 0 \end{aligned}$$

We can allow some "syntactic sugar" to make expressions like this more compact and readable, and write

$$F_{new\_edge(X,Y)} \equiv \texttt{ WIF } \mathit{edge}(Y, X) \wedge \mathit{red}(Y) \texttt{ THEN } 0.5 \texttt{ ELSE } 0,$$

with the understanding that this is not a proper extension of the representation language, but only a shorthand for expressions that are constructed according to the existing syntax rules.

The three constructs introduced so far allow to condition the probability of $R_h(\boldsymbol{i})$ on Boolean combinations of properties of $\boldsymbol{i}$ according to the relations $R_1, \ldots, R_{h-1}$. The fourth and central syntactic construct enables us to condition probabilities for $\boldsymbol{i}$ on properties of other entities $\boldsymbol{j}$. This construct requires the distinction between input relations $\mathcal{R}_{in}$ and probabilistic relations $\mathcal{R}_{prob}$ described at the end of Section 6.

Combination Functions

Assume that $F_1, \ldots, F_t$ are probability formulas.

$$F \equiv \begin{array}{l} \texttt{COMBINE } F_1, \ldots, F_t \\ \texttt{WITH } < \textit{combination function} > \\ \texttt{FORALL } < \textit{variables} > \\ \texttt{WHERE } < \textit{Boolean } \mathcal{R}_{in} \textit{ condition} > \end{array} \tag{34}$$

is a probability formula. This syntax rule is dependent on a few supplementary specifications: $<$ *variables* $>$ is simply a list of variable names. A $<$ *Boolean* $\mathcal{R}_{in}$ *condition* $>$ is a Boolean expression built from atomic expressions that can be either atoms $R(\boldsymbol{Y})$ with $R \in \mathcal{R}_{in}$, or equalities $Y = Z$ between variables (again, no identifiers for specific entities i are allowed). A $<$ *combination function* $>$, according to the original definition of [29], is any function that maps multisets of probability values $\{\!|p_1, \ldots, p_K|\!\}$ to a probability value. The most important such combination functions are

$$\textit{noisy-or}\{\!|p_1, \ldots, p_K|\!\} = 1 - \prod_{i=1}^{K}(1 - p_i) \tag{35}$$

$$\textit{mean}\{\!|p_1, \ldots, p_K|\!\} = \frac{1}{K}\sum_{i=1}^{K} p_i \tag{36}$$

One can relax the condition that both input and output values always have to be probabilities, and also allow e.g. summation:

$$\textit{sum}\{\!|p_1, \ldots, p_K|\!\} = \sum_{i=1}^{K} p_i. \tag{37}$$

However, when such constructs are used which can generate numbers outside of $[0, 1]$, then for the eventual specification of the conditional probability $P_V(R_h(\boldsymbol{i})|R_1, \ldots, R_{h-1})$ these numbers have to be brought back into the $[0, 1]$ interval. The most useful tool for this is the combination function which by a slight abuse of terminology we call the logistic-regression function:

$$\textit{logistic-regression}\{\!|p_1, \ldots, p_K|\!\} = \frac{1}{1 + \textit{exp}(-\sum_{i=1}^{K} p_i)} \tag{38}$$

Note that we have overloaded the term 'combination function' to both denote the probability formula (34), and the concrete numerical combination functions at its core. A full formal specification of the semantics of a combination function construct requires some care regarding the variables that appear in different components of the formula, and how substitutions of domain entities for these variables are performed. However, the basic principle can be described quite easily: suppose that $<$ *variables* $>\equiv Y_1, \ldots, Y_k$. Then, for a given graph $(V, R_1, \ldots, R_{h-1}, \boldsymbol{R}_{in})$, and a tuple $\boldsymbol{i}$, the $<$ *Boolean* $\mathcal{R}_{in}$ *condition* $>$ defines the set of all $\boldsymbol{j} \in V^k$ that make the condition true when one substitutes j_l for Y_l $(l = 1, \ldots, k)$, and the elements of $\boldsymbol{i}$ for other designated variables appearing in the Boolean condition. Let $J(\boldsymbol{i})$ be the set of these satisfying k-tuples of domain elements. For each $\boldsymbol{j} \in J(\boldsymbol{i})$, and each F_m $(m = 1, \ldots, t)$ the value $\textit{eval}(F_m, \boldsymbol{i}, \boldsymbol{j}, G)$ is already defined. Then the value of the formula (34) is

$$\textit{eval}(F, \boldsymbol{i}, G) = \textit{comb}\{\!|\textit{eval}(F_m, \boldsymbol{i}, \boldsymbol{j}, G)|m = 1, \ldots, t; \boldsymbol{j} \in J(\boldsymbol{i})|\!\} \tag{39}$$

where *comb* is the combination function declared in the `WITH` clause of (34).

▶ **Example 5.** Consider again the graph of Figure 4. Assume now that *edge* $\in \mathcal{R}_{in}$ is an input relation, whereas *red* $\in \mathcal{R}_{prob}$ is probabilistic. Then we can define the conditional probability for the *positive* node attribute by

$$\begin{aligned} F_{positive(X)} \equiv\ & \texttt{COMBINE}\ 0.7 \cdot red(Y) \\ & \texttt{WITH}\ noisy\text{-}or \\ & \texttt{FORALL}\ Y \\ & \texttt{WHERE}\ edge(Y, X). \end{aligned} \tag{40}$$

The product $0.7 \cdot red(Y)$ in the `COMBINE` clause here again is a syntactic shorthand for what in principle is another *product* combination function. This formula expresses a standard causal model according to which each *red* source node of an edge causes the target node to be *positive* with probability 0.7. For each $i \in \{1, \ldots, 7\}$ here $J(i)$ is just the set of nodes j with $edge(j, i)$. We obtain

$$eval(F_{positive(X)}, i, G) = \begin{cases} 0 & \text{for } i = 1, 3, 6 \\ 1 - (1 - 0.7) = 0.7 & \text{for } i = 4, 5, 7 \\ 1 - (1 - 0.7)^2 = 0.91 & \text{for } i = 2 \end{cases}$$

If we want to prevent self-loops of red nodes from being possible causes for that node to be *positive* (as happens for node 4 in the example), we can strengthen the `WHERE` clause to $edge(Y, X) \wedge Y \neq X$.

The formula is an example of a typical model for information or causal influence propagation along edge relations, and is closely related to the message passing principle in GNNs. However, combination functions are not limited to this pattern of information diffusion. If we modify the `WHERE` clause to be just the logical constant *true*, then $J(i) = \{1, \ldots, 7\}$ and $eval(F_{positive(X)}, i, G) = 1 - (1 - 0.7)^3$ for all i. Thus, the probability for $positive(i)$ now depends uniformly for all i on the global graph feature of the total number of *red* nodes. Exactly the same formula could then also be employed to define the probability for a global graph label $class()$. Going in the opposite direction, we could use still the same formula to define the probabilities for the binary *new_edge* relation (defining an Erdős-Rényi random graph model where the edge probability is a function of the number of red nodes in the graph). Generally, a probability formula with k "free variables" ($k = 1$ in (40), and $k = 0$ when the `WHERE` clause is changed to *true*) can be used to define conditional probabilities for relations of arities $\geq k$.

8 Expressivity

In Section 6 we have identified at an abstract level the similarities between what GNNs and SRL frameworks represent and compute. The high-level semantic analogies do not mean that the concrete functions that are supported by GNN or SRL models have much in common. However, already Example 5 has indicated that there are some commonalities between the message passing operations in GNNs, and probabilistic combination operations in SRL, especially RBNs. In this and the next section we will establish strong correspondences between the concrete modeling capacities of GNNs and RBNs.

The question of expressivity has been investigated for SRL frameworks [31], and also has received considerable attention in recent years for GNNs [75, 51, 4, 64, 19]. We start by looking a bit deeper into the expressivity of GNNs.

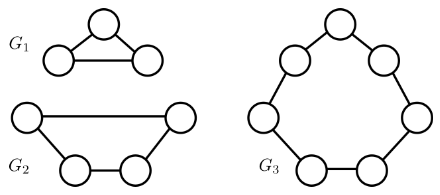


Figure 5 Indistinguishable nodes and graphs.

8.1 GNN expressivity

Broadly speaking, here expressivity relates to a GNNs capability to differentiate between different inputs. The focus can be on differentiating between different input graphs, or between nodes in graphs. In both cases, the ability to differentiate between inputs is a pre-condition for being able to support a rich class of predictive functions.

The node-level version of expressivity can be cast as the following question: for what graphs $G = (V, E), G' = (V', E')$, and nodes $i \in V, i' \in V'$, can a particular GNN architecture (or a certain class of architectures) learn representations $\boldsymbol{h}^m(i), \boldsymbol{h}^m(i')$, such that $\boldsymbol{h}^m(i) \neq \boldsymbol{h}^m(i')$? At graph level, the question becomes for which pairs of graphs G, G', the value of the readout function (9) can be different. Since (9) depends on node representations as input, the discrimination capabilities at node and graph level are tightly linked.

▶ **Example 6.** Figure 5 (adapted from [1]) shows three graphs G_1, G_2, G_3. The nodes here do not have any attributes or identifiers, so the initial representations $\boldsymbol{h}^0(i)$ would be the same constant for all nodes i in all three graphs. In a computation of $\boldsymbol{h}^1$ by any form of message-passing update (7), each node i will also obtain the same representation $\boldsymbol{h}^1(i)$, because all nodes sum identical $\boldsymbol{h}^0$ representations for exactly two neighbors j. By induction, representations $\boldsymbol{h}^k(i), \boldsymbol{h}^k(j)$ for $i \neq j$ can never become different at any step k. At the graph level, however, the three graphs could be distinguished by a final readout aggregator (9), because that would receive as input multisets of different cardinalities for the three graphs. Finally, if one considers the graph $G_4 = G_1 \cup G_2$, then this graph would no longer be distinguishable from G_3, because now (9) receives as input for the both graphs multisets of the same cardinality of identical node representations.

Many different approaches have been proposed to make GNNs more expressive than what can be achieved by the basic form of message passing (21) that we assumed in Example 6. One possible strategy is to consider node identifiers: it is clear that when unique node identifiers are used as initial representations, then already the initial representation distinguishes all nodes, and our expressivity question at the node level becomes moot (though graph-level discrimination is not immediately solved by node identifiers). However, as discussed in Section 6, the use of node identifiers would severely limit inductive generalization capabilities of models that depend on them (cf. Figure 3). A few papers have studied the use of randomly generated initial attributes as a means to combine some benefits of identifiers with (still somewhat limited) generalization abilities [65, 1]. A full review of these approaches is beyond the scope of this article. However, the following example (inspired by [65]) illustrates the main traits of these approaches.

▶ **Example 7.** Consider again the graphs in Figure 5. We shall see that by assigning random initial node attributes, we can construct a GNN, which otherwise follows the simple architecture (21), that can identify nodes that lie on a cycle of length 3, and hence can distinguish the nodes in G_1 from the nodes in G_2 and G_3. Due to the probabilistic nature of the construction, this will only be guaranteed with a certain probability $1 - \delta$ that can be brought arbitrarily close to 1.

Let N be an integer that one should think of as being significantly larger than the cardinalities n of our input graphs. For a graph $G = (V, E)$, and node $i \in V$ we generate a random initial N-dimensional $\boldsymbol{h}^0(i)$ in the form of a random one-hot vector (i.e., $\boldsymbol{h}^0(i)$ has a 1 in one randomly chosen position, and 0s everywhere else). Let $idx(i) \in 1, \ldots, N$ denote the index at which $\boldsymbol{h}^0(i)$ is 1. For a given $\delta > 0$ we can choose an N, such that for graphs with $|V| \leq n$ with probability at least $1 - \delta$ the $idx(i)$ are different for all nodes $i \in V$. The $\boldsymbol{h}^0(i)$ then can be seen as random node identifiers. For $k = 1, 2, 3$ let the dimension of $\boldsymbol{h}^k$ be $2N$. We use the first N components of these representations to just copy the initial random identifiers $\boldsymbol{h}^0$. The last N components are used to represent which nodes are reachable by a path of length k. This can be accomplished by functions of the form (21) as follows: for $k = 1$ let $\boldsymbol{W}^0$ be the $2N \times N$ matrix that consists of an $N \times N$ identity matrix in the upper half, and is zero in the lower half. Similarly, $\boldsymbol{U}^0$ is the $2N \times N$ matrix that has the identity matrix in the lower half. With $\boldsymbol{b}^0 = \boldsymbol{0}$ and f the identity function, then $\boldsymbol{h}^1(i)$ will contain a copy of i's random initial one-hot vector in the first N components, and $\boldsymbol{h}^1(i)[N + idx(j)] = 1$ iff $(i, j) \in E$, i.e., j is reachable from i by a path of length 1. The construction for $k = 2, 3$ is almost the same, with minor modifications: the matrices $\boldsymbol{W}^{k-1}, \boldsymbol{U}^{k-1}$ are now $2N \times 2N$ matrices that have $N \times N$ identity matrices in the upper left and lower right quadrant, respectively. The argument vector of $f()$ in (21) can now have integer values > 1 in some of the components $N + idx(j)$ if there exist multiple paths from i to j. This can be brought back to a pure 0/1-valued indicator vector for the existence of paths by using for f the truncated Relu function $f(x) = min(1, Relu(x))$.

Node i now lies on a cycle of length 3 iff $\boldsymbol{h}^3(i)$ has a 1 both in components $idx(i)$ and $N + idx(i)$ (i.e. i is reachable from itself by a path of length 3). Defining $\boldsymbol{h}^4(i) = Relu(\boldsymbol{h}^3(i)[1 : N] - \boldsymbol{h}^3(i)[N + 1 : 2N] - \boldsymbol{1}) \in \mathbb{R}^N$ (which still fits the functional form (21)) then gives an N-dimensional representation of i that has a single 1 in component $idx(i)$ if i lies on a cycle of length 3, and is 0 everywhere otherwise. A final summation $\boldsymbol{h}^5(i) = \sum_{h=1}^{N} \boldsymbol{h}^4(i)[h]$ then gives a scalar that classifies i as lying on a length 3 cycle or not.

Example 7 is quite representative of the general results of [65, 1] in that:

- the random initial features are exploited by otherwise standard GNN architectures;
- high-dimensional representations $\boldsymbol{h}^k$ are required;
- for a given level $1 - \delta$ of confidence in the correctness of the outcomes, the set of possible inputs has to be constrained, and thus the inductive generalization capabilities are limited.

Several other approaches have been proposed for increasing the expressivity of GNNs:

- the use of *higher order* GNNs in which representations are not associated with single nodes, but with tuples or sets of nodes [51, 72].
- more sophisticated functions than simple summation as in (21) for aggregating representations of neighbor nodes. These functions ideally are injective, i.e., map distinct multiset inputs to distinct outputs, and thereby preserve discriminative information provided by graph neighbors to the highest possible extent [75]. We return to this in Section 11.2.

8.2 The ACR architecture and first-order logic

A relatively simple approach to increase the expressivity of the basic message passing architecture (7) was proposed in [4], based on the observation that with (7) node representations $\boldsymbol{h}^k(i)$ are limited to information that is visible within a k-hop neighborhood of i. This can easily be remedied by already allowing in the computation of node representations global readout aggregations (9). The abstract representation update function then becomes

$$\boldsymbol{h}^{k+1}(i) = F^k(\boldsymbol{h}^k(i), \{\!\!\{\boldsymbol{h}^k(j) | j \in N_i\}\!\!\}, \{\!\!\{\boldsymbol{h}^k(j) | j \in V\}\!\!\}, G), \tag{41}$$

which can be instantiated to a concrete form analogous to (21):

$$\boldsymbol{h}^{k+1}(i) = f\left(\boldsymbol{W}^k\boldsymbol{h}^k(i) + \boldsymbol{U}^k \sum_{j\in N_i} \boldsymbol{h}^k(j) + \boldsymbol{R}^k \sum_{j\in v} \boldsymbol{h}^k(j) + \boldsymbol{b}^k\right). \tag{42}$$

For the resulting *aggregate-combine-readout (ACR)* GNN architecture, [4] then derive an expressivity analysis using first-order predicate logic. We shall here not give a full review of first-order logic (FOL) (standard references are [63, Chapter 8], [16]), but only illustrate the main issues by examples.

A first-order formula $\phi(X)$ with one *free variable* X can define properties of nodes in a graph. For example, the formula

$$\phi(X) \equiv \exists Y_1, Y_2, Y_3 : (E(X, Y_1) \wedge E(X, Y_2) \wedge E(X, Y_3) \wedge \neg Y_1 = Y_2 \wedge \neg Y_1 = Y_3 \wedge \neg Y_2 = Y_3) \tag{43}$$

says that X is connected to three nodes Y_1, Y_2, Y_3 that are all different, i.e., X has a degree of at least 3. Assuming that the nodes in the graph have color attributes *red*,*green* and *blue* (also allowing that several of these attributes are true at the same time for a single node), then

$$\phi(X) \equiv \mathit{blue}(X) \wedge \exists Y : \mathit{red}(Y) \tag{44}$$

says that X is blue, and there exists at least one node Y that is red. The *two-variable fragment* of FOL, denoted FOL_2, consists of all formulas that contain at most 2 distinct variables. Thus, (44) belongs to FOL_2, while (43) does not. An extension of the syntax of FOL is by *counting quantifiers* $\exists^{\geq k}$ that directly state that there exist at least k different entities with a certain property. Using counting quantifiers, one can rephrase (43) as

$$\phi(X) \equiv \exists^{\geq 3} Y : E(X, Y). \tag{45}$$

This formula is equivalent to (43), but now it only makes use of two distinct variables: (45) is an element of the two-variable fragment with counting quantifiers, denoted FOLC_2.

First-order logic and each of its fragments or extensions has a certain ability to discriminate nodes in a graph. Specifically, consider the set of graphs $\mathcal{G}(\cdot, \mathcal{E}, \mathcal{A}) := \cup_V \mathcal{G}(V, \mathcal{E}, \mathcal{A})$ where the signature $\mathcal{A}$ only contains Boolean attributes. A *(Boolean) node property* for this set of graphs is a mapping ρ that takes a graph $G = (V, E, \boldsymbol{A}) \in \mathcal{G}(\cdot, \mathcal{E}, \mathcal{A})$ and a node $i \in V$ as input, and returns 0 or 1. A node property is captured by a logic formula $\phi(X)$ if $\rho(G, i) = 1$ iff $\phi(X)$ evaluates to true for $X = i$. A central result of [4] then is

▶ **Theorem 8** ([4, Theorem 5.1]). *If a node property ρ is captured by a formula in* FOLC_2*, then ρ can be computed by an ACR-GNN of the form (42) with f the truncated Relu activation function, and the node attribute vectors $\boldsymbol{A}[i, \bullet]$ as initial representations.*

This result is remarkably similar to an expressivity result for RBNs given in [29]. Adapted to our current context, that result can be stated as

▶ **Theorem 9** ([29, Theorem 1]). *If a node property ρ is captured by a formula in FOL, then there exists a probability formula $F_{\rho(X)}$ that only uses the noisy-or combination function, such that* $\mathrm{eval}(F_{\rho(X)}, i, G) = \rho(G, i)$.

The proof of Theorem 8 is non-trivial, and depends on an alternative characterization of FOLC_2 as a special modal logic. The proof of Theorem 9, on the other hand, is straightforward, as the construction of a probability formula corresponding to a given FOL formula ϕ can simply follow the structure of ϕ, using wif-then-else constructs to capture Boolean operations, and noisy-or combination functions to capture existential quantification.

Theorem 9 is somewhat stronger than Theorem 8, as FOL is more expressive than FOLC_2. Moreover, the original theorem of [29] is more general than what is stated in Theorem 9, as beyond node properties it also covers properties of whole graphs, and of k-tuples ($k \geq 2$) of nodes. The ability to express features of k-tuples of nodes via probability formulas with k free variables is key for the high flexibility and expressivity of RBNs and many other SRL frameworks (using, of course, somewhat different representation techniques than probability formulas). This also ensures that probability formulas are still more powerful than higher order GNNs mentioned above.

Theoretical expressivity analyses are mostly based on classes of properties that can be expressed in a formal framework, such as logic characterizations that we have focused on here, or classes of *Weisfeiler-Lehman (WL)* graph isomorphism tests, which have played a central role in the expressivity analysis of GNNs [64]. However, in reality a GNN or SRL model will rather need to represent complex noisy relationships, not clear-cut logical properties. In the next section we will show that RBNs can also represent all functions that do not represent logic properties, and which can be represented by standard GNN architectures.

9 RBN encodings of GNNs

In this section we show how an ACR-GNN composed of layers of the form (42) can be encoded as a probability formula as introduced in Section 7. Let $\mathcal{N}$ be an ACR-GNN defined by matrices/vectors $\boldsymbol{W}^k, \boldsymbol{U}^k, \boldsymbol{R}^k, \boldsymbol{b}^k$ $(k = 1, \ldots, m)$, as well as a final output (8) or readout (9) layer. Assume, for now, that the function f in (42) is the sigmoid activation function. Also assume that all node attributes $\mathcal{A}$ are Boolean, represented by one-hot encodings in $\boldsymbol{h}^0$. We show that for each k, and each $l = 1, \ldots, n_k$, there exists a probability formula $F_{\boldsymbol{h}^k[l]}(X)$, such that for all attributed graphs $G = (V, E, \boldsymbol{A})$, and all $i \in V$:

$$\boldsymbol{h}^k(i)[l] = \mathit{eval}(F_{\boldsymbol{h}^k[l]}, i, G). \tag{46}$$

First consider $k = 0$ and $1 \leq l \leq n_0$. Then there exists an attribute $A \in \mathcal{A}$, and a truth value $\tau \in \{\mathit{true}, \mathit{false}\}$, such that $\boldsymbol{h}^0(i)[l]$ is the 0,1-valued indicator for whether node i has value τ for A. For our Boolean attributes A this somewhat redundant encoding could obviously be reduced to a single 0,1-valued input, using 0 for $\tau = \mathit{false}$, and 1 for $\tau = \mathit{true}$. A "mechanical" application of one-hot encodings will give us this redundant two component encoding, however. We then define

$$F_{\boldsymbol{h}^0[l]}(X) \equiv \begin{cases} A(X) & \text{if } \tau = \mathit{true} \\ \neg A(X) & \text{if } \tau = \mathit{false} \end{cases} \tag{47}$$

where $\neg A(X)$ is a shorthand for `WIF` $A(X)$ `THEN` 0 `ELSE` 1. Now assume that formulas $F_{\boldsymbol{h}^k[l]}$ have been constructed for some $k \geq 0$. We then can first define formulas that compute the two sums in (42). For the first sum over the neighbor representations, we can use

$$\begin{aligned} F_{\sum_{E(\cdot,X)} \boldsymbol{h}^k[l]}(X) \equiv\ & \texttt{COMBINE } F_{\boldsymbol{h}^k[l]}(Y) \\ & \texttt{WITH } \mathit{sum} \\ & \texttt{FORALL } Y \\ & \texttt{WHERE } E(Y,X). \end{aligned} \tag{48}$$

A similar formula $F_{\sum_V \boldsymbol{h}^k[n_k]}()$ is used to represent the second sum ranging over all nodes $j \in V$. In that formula the `WHERE` clause simply is the Boolean *true* constant, and the formula then does not depend on the node X. Let us abbreviate the first sum in (42) by $\overline{\boldsymbol{h}^k}^i$ (this one depends on i), and the second sum by $\overline{\boldsymbol{h}^k}$ (no dependence on i). Then

$$\boldsymbol{h}^{k+1}(i)[l] = f(\boldsymbol{W}^k[l,\bullet] \cdot \boldsymbol{h}^k(i) + \boldsymbol{U}^k[l,\bullet] \cdot \overline{\boldsymbol{h}^k}^i + \boldsymbol{R}^k[l,\bullet] \cdot \overline{\boldsymbol{h}^k} + \boldsymbol{b}^k[l]). \tag{49}$$

Expanding the dot products between n_k-dimensional vectors contained in this expression, we can write this as the probability formula

$$\begin{array}{lll} F_{\boldsymbol{h}^{k+1}[l]}(X) \equiv & \texttt{COMBINE} & \boldsymbol{W}^k[l,1] \cdot F_{\boldsymbol{h}^k[1]}(X), \\ & & \vdots \\ & & \boldsymbol{W}^k[l,n_k] \cdot F_{\boldsymbol{h}^k[n_k]}(X), \\ & & \boldsymbol{U}^k[l,1] \cdot F_{\sum_{E(\cdot,X)} \boldsymbol{h}^k[1]}(X), \\ & & \vdots \\ & & \boldsymbol{U}^k[l,n_k] \cdot F_{\sum_{E(\cdot,X)} \boldsymbol{h}^k[n_k]}(X), \\ & & \boldsymbol{R}^k[l,1] \cdot F_{\sum_V \boldsymbol{h}^k[1]}(), \\ & & \vdots \\ & & \boldsymbol{R}^k[l,n_k] \cdot F_{\sum_V \boldsymbol{h}^k[n_k]}(), \\ & & \boldsymbol{b}^k[l] \\ & \texttt{WITH} & \textit{logistic regression} \\ & \texttt{FORALL} & \\ & \texttt{WHERE} & \textit{true} \end{array} \tag{50}$$

The products appearing here are products of scalar quantities defined by probability formulas, and strictly speaking another shorthand for formulas of the form `WIF` F_1 `THEN` F_2 `ELSE` 0. Formula (50) is a degenerate combination function in the sense that it does not aggregate over any entities, as visible from the empty `FORALL` clause (the following `WHERE` clause then is somewhat redundant). Aggregation here only is over the fixed number of $t = 3n_k + 1$ sub-formulas. Since the *logistic-regression* combination function sums its arguments, and then applies the sigmoid function, (50) computes exactly (48), when f there is the sigmoid.

In a similar manner, also probability formulas representing the components of an output (8) or readout (9) layer can be constructed. To accommodate other activation functions, such as Relu or truncated Relu, corresponding combination functions have to be used.

In terms of representation size, the encoding of (42) by probability formulas clearly leads to a significant blow-up, as the matrix-vector multiplications are decomposed down to the level of operations on scalars. It is important to realize, however, that mathematically the encoding is faithful: the evaluation of the probability formulas leads to exactly the same basic multiplication, addition, and sigmoid application operations, as in a "forward propagation" evaluation of the neural network layers. More importantly, also gradient-descent based learning of the parameters $\boldsymbol{W}^k, \boldsymbol{U}^k, \boldsymbol{R}^k, \boldsymbol{b}^k$ using a standard algorithm like LBFGS [54] or ADAM [40] leads to exactly the same algorithmic steps, assuming that equivalent loss functions for the final output of the probability formula, respectively the ACR-GNN, are used (cf. Section 12.2).

We have here shown how GNNs can be represented as RBNs (focusing on ACR-GNNs, but similar constructions can be done for other GNN architectures). The central message-passing paradigm of GNNs can also be captured by other SRL frameworks than RBNs, especially frameworks in the Bayesian network constructor class, which all contain conditioning on relational neighbors as a central modeling tool. However, RBNs are specifically well-suited for a direct encoding of GNN architectures, because of the following features:

- The translation of symbolic to numeric data performed by the semantics (31) of atomic probability formulas directly bridges the gap between symbolic SRL and numeric GNN approaches.
- The recursive syntax definition of probability formulas directly corresponds to the "deep" structure of GNN architectures.

Obviously, just encoding a GNN as an RBN serves little purpose if one then solves identical tasks using the RBN representation, as one would solve with a GNN. Indeed, this would be a rather bad idea, because even though the RBN representation is, in principle, mathematically and algorithmically equivalent to the GNN model, it is in practice computationally much less efficient. A main reason for this is that GNNs only permit aggregation operations over a node's neighbors (or, for a readout, over all nodes in the graph), and this corresponds to simple matrix-vector multiplications involving the adjacency matrix. RBNs, on the other hand, support aggregations over all kinds of sets of tuples of nodes that can be defined with a Boolean condition in the `WHERE` clause of a combination function. The retrieval of the relevant tuples is implemented by what amounts to a general database query function. Even the simple queries '`FORALL` Y `WHERE` $E(Y,X)$' we encounter in the RBN encoding of a GNN are then a bit more involved to compute than simply retrieving row X of E's adjacency matrix. Encoding a GNN model in an RBN can be beneficial, however, if one then leverages capabilities of SRL models that are not provided by a GNN:

- Solving inference tasks other than the single prediction task for which a GNN is trained.
- Combining low-level "neural" model components with higher level symbolic representations, e.g. expressing expert domain knowledge.

We will illustrate the first point in Example 11 below. First we consider an example for ACR-GNNs in their original form, however.

▶ **Example 10.** Barceló et al. [4] considered logically defined Boolean class labels for nodes in attributed graph over a signature of color attributes $\mathcal{A} = \{blue, green, yellow, red, purple\}$. The simplest label definition considered in [4] is expressed by the FOLC$_2$ formula

$$\alpha_1(X) \equiv \exists^{[8,10]} Y(blue(Y) \wedge \neg edge(X,Y)), \tag{51}$$

where $\exists^{[8,10]}$ is shorthand for $\exists^{\geq 8} \ldots \wedge \neg\exists^{\geq 11} \ldots$. Using the property α_1 defined by this formula, a more complex property α_2 is defined by

$$\alpha_2(X) \equiv \exists^{[10,20]} Y(\alpha_1(Y) \wedge \neg edge(X,Y)). \tag{52}$$

According to Theorem 8, the node properties (51), (52) can be captured by ACR-GNNs. For (51) it is sufficient to use an ACR-GNN with a single ACR layer (42) of dimension $n_1 = 2$: the parameters $\boldsymbol{W}^1, \boldsymbol{U}^1, \boldsymbol{R}^1, \boldsymbol{b}^1$ can be set such that $\boldsymbol{h}^1(i)[0]$ becomes a 0,1-valued indicator for whether node i has at least 8 blue non-neighbors, and $\boldsymbol{h}^1(i)[1]$ indicates whether this number is no more than 10. Based on this representation, an output layer (8) can then provide an exact classification of (51). Similarly, (52) can be represented by a two-layer ACR-GNN with $n_1 = n_2 = 2$.

Table 2 ACR-GNN accuracies for α_2.

	$n_i =$				
m	2	4	16	64	128
1	0.756|0.407	0.698|0.697	0.830|0.713	0.831|0.716	0.833|0.714
2	0.696|0.502	0.683|0.649	0.833|0.704	0.891|0.886	0.869|0.795

Using the ACR-GNN implementation provided by the authors of [4] [1] we can re-create and extend some of their experiments. Table 2 shows the accuracies for the property α_2 that are obtained by ACR-GNN models trained on randomly generated graphs of sizes $40 \leq n \leq 50$. To test the generalization capabilities of the learned models, they are tested on graphs in the same size range, and also on a test set of slightly larger graphs with sizes ranging in $51 \leq n \leq 60$. In Table 2 the accuracies for these two different test sets are shown in the format $<$ *small graph accuracy* $>$ | $<$ *large graph accuracy* $>$. In our experiment we vary the number of layers $m = 1$ or $m = 2$, and their dimensions n_i (for $m = 2$, always $n_1 = n_2$). The base frequency of nodes with the α_2 property is 0.643 and 0.396 for the small and large graphs test sets, respectively. One can see that even though the architecture with $m = 2$ and $n_i = 2$ is sufficient to capture α_2 in principle, the stochastic gradient optimization here does not succeed to construct a model with an accuracy that is notably better than a baseline predictor. To obtain higher accuracies, a significant over-parameterization is required ($n_i = 64$ corresponds to the experimental setting of [4]). We will return to the benefits of over-parameterization in neural network learning in Section 12.3. For the simpler property α_1, a similar experiment leads to perfect accuracies of 1.0 in all settings, and for both test sets.

▶ **Example 11.** We now consider RBN models for the simpler target α_1 (51). We consider three different models:

- RBN-manual: a manually designed RBN that directly encodes the logical formula α_1 following the construction behind Theorem 9.
- RBN-gnn-learned: an RBN encoding of an ACR-GNN with one hidden layer of dimension 4, following the construction described in Section 9. As described in Example 10, this is (more than) enough to represent a precise model for α_1. Parameters learned from training data using stochastic gradient descent.
- RBN-gnn-manual: the same RBN structure as the previous, but parameters set manually to encode α_1 (most of the manually set parameters are zeros, since only two of the four dimensions are actually needed).

For all three models we can consider a *discriminative* model where the attribute *blue* is assumed to be a fixed input (as it would be for a GNN model), and a *generative* version where we also take *blue* to be probabilistic. This is easily effected by adding to the RBN a very simple formula

$$F_{blue(X)} \equiv 0.26$$

that specifies the marginal probability for a node to be blue. The value 0.26 is the empirical probability of *blue* in the training data, which in our two manual models is set manually, and in the learned model is learned jointly with the other model parameters. Note that the *edge* relation still is assumed to be given as input, so we still do not have a fully generative model for graphs, but only a conditional model for the attributes given the graph structure.

[1] `https://github.com/juanpablos/GNN-logic`

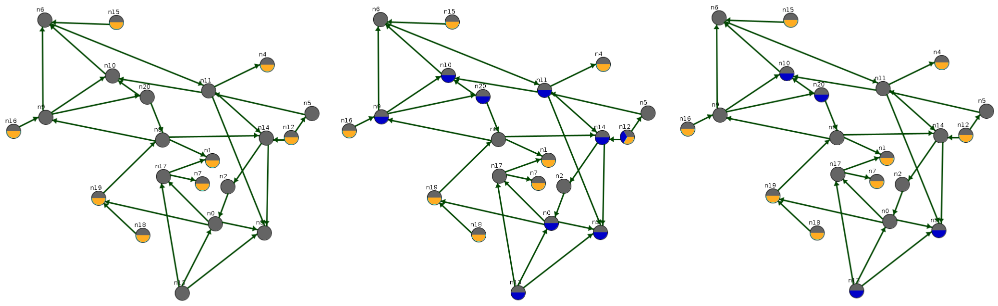


Figure 6 Most probable explanations in Example 11.

Evaluated on two test sets with different graph sizes, as described in Example 10, all three discriminative models achieve an accuracy of 1.0 for both test sets (recall that the same was observed for ACR-GNNs on the simple α_1 target). Using the generative models, we now can also consider queries beyond the prediction of α_1. One interesting type of query is to turn the role of input attribute *blue* and target α_1 around, and assume that given observed α_1 labels, we want to predict the (now) unobserved attribute *blue*. One particularly interesting version of this question is to ask for the *most probable explanation* (MPE) of the observed α_1 values, i.e. to find the assignment of *blue* attributes that makes the observed α_1's most probable.

Figure 6 on the left shows a small graph with 21 nodes where nodes with the observed α_1 label are marked by an orange color segment. Edges here are shown as directed, but this direction is ignored by the models. From any of our generative models we can now calculate an MPE assignment vector $\boldsymbol{b}$ for the *blue* attribute that maximizes the probability $P_V(\boldsymbol{\alpha}_1|\boldsymbol{b})$ (this is a non-trivial optimization problem, which is tractable for this small example). Figure 6 in the middle shows the MPE assignment of *blue* that is obtained from either of the two manual RBN models. In this explanation, there is a total of 9 blue nodes, and nodes have the α_1 label iff they are connected to at most one of these blue nodes, i.e., (51) holds. Performing MPE inference for the RBN-gnn-learned model yields the assignment of the *blue* attribute shown in the right graph of Figure 6. This explanation is inconsistent with the logical definition of the α_1 label, since here there are only 4 blue nodes in the domain, which implies that α_1 should be false for all nodes, contrary to the given observation. This indicates that the learned model does not generalize to this MPE inference task.

The reason for this failure of the learned model could be twofold: 1) the model, even though perfectly accurate on test sets that are very similar or only slightly larger than the training examples, does not generalize to our MPE query graph, which is smaller than the training examples, and may also exhibit different connectivity patterns of the *edge* relation; 2) the training objective is not appropriate for solving MPE tasks. Turning to point 2) first: as we will show in Section 12.2 below, the objective function we use for training the RBN is equivalent to the classification loss function used for training an ACR-GNN classifier. However, both are equivalent to maximizing the log-likelihood, which can be seen as a "universal" objective for learning a generative probabilistic model for all probabilistic inference tasks. This means that even if right from the beginning we had intended our model to be used for MPE inference as considered here, our training objective would not have been different. That indeed point 1) seems to be the issue is revealed when we now consider our original classification task for the graph of Figure 6: we use this graph with the *blue* attribute set as in the middle graph of Figure 6. For this input graph the learned model only achieves an accuracy of 0.47 for classifying the α_1 label for the 21 nodes. Thus, the learned model does not capture the logical nature (51) of the target well enough in order to generalize to input graphs that are rather different in size (and possibly structure) from the training examples.

Lastly, we consider an RBN model that combines domain knowledge with learning: suppose it is known that the α_1 label depends on the number of blue non-neighbors, but the exact bounds $[8, 10]$ in (51) are unknown. Our qualitative knowledge then is captured by the logical form of the probability formulas in RBN-manual. The missing quantitative information corresponds to unknown values of the numeric constants in these formulas. Learning these parameters gives us a model RBN-manual-learned with manually defined structure and learned parameters. This model then turns out to perform perfectly well both on our classification and MPE tasks.

10 Dealing with Homophily

A key property in network data is the phenomenon of *homophily*: connected entities tend to be similar. This phenomenon is particularly well documented for social networks, where, e.g., a *friendship* relation between two people is indicative of similar political leanings, social status, and other properties (cf. (13)). Similar in other types of networks: in bibliographic networks, papers citing each other are likely to be in the same subject area. In sensor or traffic networks, entities that are connected by a spatial proximity relation tend to have similar properties. When a given class label exhibits homophily, it will be important to exploit this for classification. Taking homophily into account has two different aspects:

- Collective classification: the prediction of class labels for entities should be done jointly for all (unlabeled) entities, such that the joint labeling exhibits homophily.
- Autoregression: the prediction of a class label depends on observed labels for some entities.

These two aspects are non-exclusive, but distinct. For illustration, consider the two small graphs in Figure 7. In the graph on the left an attribute with possible values 'yellow' and 'blue' is observed for all nodes, whereas the class label with values 'red' and 'black' is unobserved for all nodes. In the graph on the right, the class label is observed for three nodes. Assuming that the class label exhibits homophily (maybe this is learned from some other labeled training graphs), one would want to assign to all the nodes belonging to one of the two cliques in the graph the same label. This would be a case of collective classification, where the label assigned to one node constrains what label we assign to other nodes. In the graph on the right, homophily would indicate that the nodes in the clique at the top should be labeled 'red', and the nodes in the bottom clique be labeled 'black'. This would then require an autoregressive component in the classification. Homophily mostly plays a role in transductive problem settings, as illustrated by the graph on the right.

Among SRL frameworks, the Markov network constructor and probabilistic logic programming types are able to model homophily most easily. For Markov logic networks an example is already given by (13): the undirected nature of these logical feature specifications and the Markov network semantics fit very well mutual, symmetric dependencies between attributes due to homophily. Probabilistic logic programs can represent autocorrelation via clauses like

$$\mathit{republican}(X) \leftarrow \mathit{friends}(X, Y) \wedge \mathit{republican}(Y).$$

The least fixed-point semantics of the logic program then allows to propagate the *republican* label from some observed republicans to other unlabeled entities (the probabilistic component of the program would make this propagation probabilistic). For SRL frameworks of the

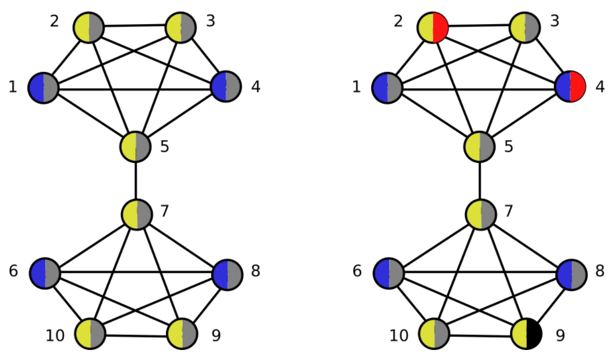


Figure 7 Homophily challenge.

Bayesian network constructor type the required acyclicity of probabilistic dependencies is a certain hurdle for a direct modeling of homophily. We will show below how this hurdle can be overcome.

GNNs encounter inherent challenges for dealing with homophily. A GNN for predicting a node class label is relying on node features other than the label for making this prediction. If nodes are indistinguishable based on the available features, they can only be assigned the same label (cf. Section 8). For our example in Figure 7 on the right this means that a predictive model for the class label that only is a function of the observed *color* attribute and *edge* relation must give identical labels to the nodes in the sets $\{1, 4, 6, 8\}$, $\{5, 7\}$ and $\{2, 3, 9, 10\}$, respectively, because nodes in these sets are pairwise indistinguishable based on these two relations. This limitation of GNNs has motivated the combination of GNNs with Markov logic networks in [59].

A standard strategy in probabilistic modeling for representing dependencies that cannot be explained by observed features is the introduction of *latent variables*. For RBNs, modeling with latent numeric relations has been introduced in [36]. Originally mostly motivated by applications in community detection, the same technique also applies to classification under homophily.

▶ **Example 12.** A numeric k-ary relation r simply is an assignment of a real number to ground atoms $r(i_1, \ldots, i_k)$ $(i_j \in V)$. Numeric relations can represent actual observable numeric data, such as numeric node attributes or edge weights. We use numeric relations as latent features that are not observed, and that are not part of the generative model. For the simple scenario as depicted in Figure 7 we may assume that both the *edge* relation and the class label depend on an unobserved node feature that determines both the propensity of nodes to connect by an edge, and the likely value of their class label. Representing this node feature by a latent numeric attribute $latent(X)$, we can model the *edge* and *class* relation by the two formulas

$$\begin{aligned} F_{class(X)} &\equiv \texttt{COMBINE}\ latent(X)\ \ \texttt{WITH}\ logistic\ regression \\ F_{edge(X,Y)} &\equiv \texttt{COMBINE}\ latent(X) \cdot latent(Y)\ \ \texttt{WITH}\ logistic\ regression \end{aligned} \tag{53}$$

Assuming that all $i \in V$ are assigned a value $latent(i) \in \mathbb{R}$, the probability for $class(i)$ then simply is $1/(1 + e^{-latent(i)})$, and the probability for $edge(i, j)$ is $1/(1 + e^{-(latent(i) \cdot latent(j))})$ (we omit the *color* attribute here, since it is not instrumental for the prediction of the class label). The extremely simplistic model (53) could be refined by allowing more than one latent attribute (i.e., a latent feature vector, rather than a latent scalar value), and refining the functions that map latent feature values to probabilities.

Table 3 Latent feature values and predicted probabilities from model (53).

i	1	2	3	4	5	6	7	8	9	10
$latent(i)$	1.49	1.52	1.49	1.52	1.15	−1.49	−1.15	−1.49	−1.52	−1.49
$P(class(i) = red)$	0.81	[1.0]	0.81	[1.0]	0.75	0.18	0.24	0.18	[0.0]	0.18

Given observed *edge* and *class* data, one can learn values for the *latent* attribute that best explain the data based on the maximum likelihood principle (cf. Section 12.2 below). Importantly, learning latent numerical relations in this manner is treated just as part of the general parameter learning problem, and does not require any special purpose algorithms. Learning the latent values and predicting the class labels for the 10 nodes gives the result shown in Table 3. We see that nodes 1-5 in the upper clique are clearly classified as red, and those in the lower clique as black.

Somewhat analogous to the latent variable modeling approach we have taken here within the RBN setting are the *auto-encoders* in neural networks, and especially graph auto-encoders for graph data [42]. Graph auto-encoders also construct latent feature representations of nodes that are calibrated to explain observed edges with a function corresponding to our $F_{edge(X,Y)}$ in (53) (this type of model actually has a longer history in statistical graph analysis [26]). Originally proposed in the context of link prediction tasks, graph autoencoders could also be a basis for node classification under homophily.

11 Aggregation

11.1 Invariance and Sum Aggregation

The core element both in GNN and SRL models of relational data is the aggregation of features of neighboring nodes. Since there is no defined order on the neighbors of a node, such an aggregation should not be based on any assumed ordering. In our definitions, such as (7) for GNNs, and the definition for combination functions in Section 7 for RBNs, this has been taken into account by specifying that the aggregation must be a function of a representation of the neighbors as a multiset. Aggregation by summation as in (21) is consistent with such a multiset view of the inputs, because the sum does not depend on the order of summation. In practice, however, graphs are usually represented by adjacency matrices or adjacency lists that induce an (arbitrary) order on the nodes.

In the case of GNNs, matrix and vector representations of graphs and neighbor lists are often assumed in the basic definitions of the models (e.g. [41, 79, 74], whereas e.g. [24] uses set notation in the model specification). The requirement that the implicit ordering of nodes induced by these representations does not affect the results is then often just implicitly taken into account by only considering order independent operations like summation, or the maximum or minimum operators. A more explicit and systematic consideration of admissible aggregation operations on vector representations of multisets has been initiated in [77]. We here briefly review these initial results. Reformulating the definitions of [77] slightly, we formalize our requirement as follows.

▶ **Definition 13.** *Let $\mathcal{X}$ be a set (to be thought of as a set of possible feature values). A function*

$$f : \bigcup_{n \in \mathbb{N}} \mathcal{X}^n \to \mathbb{R}$$

is called an $\mathcal{X}$-tuple aggregator*. The function f is* permutation invariant*, if for all n and all permutations π of $\{1,\ldots,n\}$, and all $x_1,\ldots,x_n \in \mathcal{X}$*

$$f(x_1,\ldots,x_n) = f(x_{\pi(1)},\ldots,x_{\pi(n)}) \tag{54}$$

holds.

We note that regardless of the nature of $\mathcal{X}$, the value of the aggregator here is required to be a real number. This is somewhat contrary to the idea that an aggregator of values from $\mathcal{X}$ should return another value in $\mathcal{X}$. However, often $\mathcal{X}$ will itself just be $\mathbb{R}$, or a subset of $\mathbb{R}$, in which case the return value of f lies in the same space as the values it aggregates. The following is then proposed as Theorem 2 in [77], and here given in a generalized form as described in Appendix A.3 in [73].

▶ **Theorem 14** ([77, 73])**.** *Let $\mathcal{X}$ be countable. Then an $\mathcal{X}$-tuple* aggregator *f is permutation invariant iff there exist functions $\phi : \mathcal{X} \to \mathbb{R}$ and $\rho : \mathbb{R} \to \mathbb{R}$, such that*

$$f(x_1,\ldots,x_n) = \rho(\sum_i \phi(x_i)). \tag{55}$$

The restriction of this theorem to countable $\mathcal{X}$ has caused some concern already in [77], and amplified in [73]. However, countability of $\mathcal{X}$ per se is not a major problem: in reality, initial (node) features will be categorical attributes (or one-hot encodings thereof), or finite precision numerical attributes. Thus, a countable $\mathcal{X}$, in principle, is sufficient to represent such initial features. Importantly, then, if $\mathcal{X}$ is countable, then so is $\bigcup_{n\in\mathbb{N}} \mathcal{X}^n$, and hence the range of f in $\mathbb{R}$. This means that countability of the feature space $\mathcal{X}_k$ at the kth GNN layer will be guaranteed, if the input feature space is countable.

The implications of Theorem 14 have sometimes been overstated: by appealing to universal function approximation properties of neural networks [28] it is suggested that it is sufficient to use summation for aggregation, in combination with additional perceptron layers that are trained to implement ρ and ϕ (e.g.[23, Section 5.2.2], [75]). However, as already pointed out in [73], the function approximation properties of neural networks are not sufficient in this case. To illustrate this point, we here reproduce the proof of Theorem 14 as given in [73].

Proof of Theorem 14. Let $x_1, x_2, \ldots$ be an enumeration of $\mathcal{X}$. Let $p_1, p_2, \ldots$ be an enumeration of all prime numbers. Define $\phi(x_i) := -\log p_i$. Then, by the unique prime factorization property of the integers, for any tuple $\boldsymbol{x} = (x_{i_1},\ldots,x_{i_n}) \in \mathcal{X}^n$ the sum $\sum_{j=1}^n \phi(x_i) = \log \frac{1}{\prod_{j=1}^n p_{i_j}}$ is an $\mathbb{R}$-valued code $\Phi(\boldsymbol{x})$ for $\boldsymbol{x}$, such that the encoding Φ is injective on $\bigcup_{n\in\mathbb{N}} \mathcal{X}^n$, and hence invertible. One can now simply define $\rho(r) := f(\Phi^{-1}(r))$ to obtain a representation of f in the form (55). ◀

The functions ϕ, ρ as constructed in this proof are outside the scope of the universal representation theorems for neural networks, which only apply to functions that are defined on a compact subset of the reals, and are continuous. To even be able to consider continuity in the sense of the representation theorems, the function ϕ would need to be defined on $\mathbb{R}$ (or $\mathbb{R}^m$, for some $m > 1$), not on $\mathcal{X}$. This, however, can be overcome by assuming (without much loss of generality) that $\mathcal{X} \subset \mathbb{R}$, and that ϕ actually is defined on all of $\mathbb{R}$. However, then an extension to $\mathbb{R}$ of ϕ as constructed in the proof can either not be continuous (if $\mathcal{X}$ has an accumulation point in $\mathbb{R}$), or not be limited to a compact set (if $\mathcal{X}$ is unbounded in $\mathbb{R}$). Due to these limitations of Theorem 14 there is still a need to consider aggregators outside the class defined by (55). Possibilities include the use of a fixed selection of standard aggregators [10], or the use of a parametric family of aggregators whose trainable parameters can be optimized to learn customized aggregation functions for each learning task [55].

11.2 Injectivity and Expressivity

The key element of the proof of Theorem 14 is the construction of an injective function $\Phi(\boldsymbol{x})$ on the space of multisets $\boldsymbol{x}$. The problem of constructing injective functions on multisets of feature values has been a key element in the study of the expressiveness of GNNs, also outside the context of investigating the universality properties of sum-aggregation (e.g.[75, 64]). An injective aggregation function would allow to preserve the full information of a node's neighbors' features $\{\!|\boldsymbol{h}^k(j)|j \in N_i|\!\}$ in the updated node representation $\boldsymbol{h}^{k+1}(i)$, and thereby enable the construction of maximally discriminative node (or graph) classifiers. However, classic results in mathematics impose strict limits to the endeavor of implementing injective aggregation functions using continuous functions as provided by neural network layers. For our purpose, we can formulate this as follows:

▶ **Theorem 15** ([8, 70]). *There does not exist a continuous injective function from $\mathbb{R}^n$ to $\mathbb{R}^m$ if $n > m$.*

In our context, n in this theorem would be the cardinality of a multiset of real numbers, and usually $m = 1$ as the target dimension for the aggregator. We now have to be a little bit careful, since our desired permutation invariance (54) actually says that we want our aggregator not to be injective on $\mathbb{R}^n$, but only to return distinct values for different multisets. We can use ordered vectors as unique representatives for multisets: defining

$$\mathbb{R}^{n,\leq} := \{(r_1, \ldots, r_n) \in \mathbb{R}^n | r_i \leq r_{i+1}, i = 1, \ldots, n-1\}, \tag{56}$$

we obtain a one-to-one correspondence of multisets of cardinality n and $\mathbb{R}^{n,\leq}$. We can now re-state Theorem 15 as

▶ **Theorem 16** ([8, 70]). *There does not exist a continuous injective function from $\mathbb{R}^{n,\leq}$ to $\mathbb{R}^m$ if $n > m$.*

This theorem imposes limits on the possibility of constructing general and expressive aggregation function for the feature space $\mathcal{X} = \mathbb{R}$ already in the case of a fixed cardinality of multisets. However, the theorem does not exclude the possibility of continuous functions on $\mathbb{R}^{n,\leq}$ that are injective on a countable subset $\mathcal{X}^n \subset \mathbb{R}^n$, which, as argued above, may be all we need. In fact, as the following example shows, this can be done.

▶ **Example 17.** Let $\mathcal{X} = \mathbb{N}$. We construct a function $f : \cup_{n\in\mathbb{N}}\mathbb{R}^n \rightarrow \mathbb{R}$, such that:
(i) the restrictions of f to arguments of fixed dimensions n are continuous;
(ii) f is permutation invariant;
(iii) f is injective for multisets of values from $\mathcal{X}$.
We first map vectors in $\mathbb{R}$ to their ordered representatives in $\mathbb{R}^{n,\leq}$:

$$f_{ord} : \cup_{n\in\mathbb{N}}\mathbb{R}^n \rightarrow \cup_{n\in\mathbb{N}}\mathbb{R}^{n,\leq}.$$

In procedural terms, $f_{ord}(\boldsymbol{r})$ is the application of a sorting algorithm to $\boldsymbol{r} \in \mathbb{R}^n$. Seen as mapping from $\mathbb{R}^n$ to $\mathbb{R}^{n,\leq}$ this is a function that satisfies properties (i)–(iii). In fact, f_{ord} is not only continuous, but also differentiable. For vectors in $\boldsymbol{r} \in \mathbb{R}^{n,\leq}$ we now define

$$f_{prime} : \boldsymbol{r} \mapsto \prod_{j=1}^{n} p_j^{r_j},$$

where, as in the proof of Theorem 14, p_j is the j'th prime number. As in that proof, the unique prime factorization of integers implies that the restriction of f_{prime} to $\cup_{n\in\mathbb{N}}\mathcal{X}^{n,\leq}$ is injective. Since f_{prime} also is continuous, we then obtain that $f = f_{prime} \circ f_{ord}$ satisfies (i)–(iii).

Table 4 Learning in GNN and SRL: a summary of correspondences.

		GNN	SRL
Structure	Space	NN architectures	(Logical) model structure
	Manual specification by	NN engineers	SRL experts, domain experts
	Learned by	Optimization/search in combinatorial spaces	
Parameters	Space	High-dimensional	Low-dimensional
	Manual specification	Never	Possible
	Objective	Loss function (cross-entropy, MSE, ...)	Likelihood (plain, conditional, pseudo-, ...)

In theoretical terms, the function f we have constructed has all the properties one could desire. In addition to (i)-(iii) it also has the important property that for $\boldsymbol{r} \in \mathbb{N}^n$ the value $f(\boldsymbol{r})$ is again an element of $\mathbb{N}$, so that the same f can be used over multiple iterations of aggregation. The requirement that the initial input features are in $\mathbb{N}$ is not a serious (theoretical) limitation, since any countable $\mathcal{X}$ could be mapped into an integer encoding. Note that a mapping of an initial input feature space $\mathcal{X}$ to an integer encoding is not subject to the continuity concerns that we otherwise have, since it can be implemented as a data preprocessing step, and need not be computed by internal (continuous) neural network functions.

In practical terms, however, f is unmanageable, due to the very large numbers produced by f_{prime}, which would soon cause numeric overflow in an implementation. Furthermore, f_{prime} again does not fulfill the requirements of the universal approximation results for neural networks. It appears to be an open question whether a function f with (i)-(iii) can be constructed that is numerically manageable, such that it can be approximated by standard neural architectures.

Theoretical questions about permutation invariance, canonical forms (based on summation as the core aggregation step), and expressivity of aggregation functions that have arisen in the field of GNNs have not been considered previously in analogous lines of investigation in SRL, even though aggregating (or combining) information from related entities also is a core element of SRL modeling. There are several reasons for this: first, permutation invariance only becomes an issue when one represents graphs by adjacency matrices. The logic-based background of SRL, and the associated “possible worlds” view of multi-relational graphs, favors a representation of graphs as the set of ground atoms that are true. When, in this manner, all fundamental definitions about syntax and semantics of SRL models are based on sets (or multisets) rather than matrices and vectors, permutation invariance never becomes an issue. The characterization of general, canonical forms of aggregation, however, would still be of interest at least for those SRL frameworks that include explicit aggregation or combination operators: these are most, if not all, of the frameworks that fall into the Bayesian network constructor category, as exemplified by the combination function construct in RBNs. Markov network constructors and probabilistic logic programming approaches, on the other hand, perform aggregation more implicitly through a single, fixed multiplicative mechanism (based on products, rather than sums as in (55).

12 Parameter and Structure Learning

In this section we discuss the role of structure and parameters in SRL and GNN learning. A summary of some of the correspondences we find is given in Table 4.

12.1 Structure

An SRL model consists of a "structure" that is given by dependency relations expressed using a logic-based or graphical representation, and numeric parameters that are needed to quantitatively define a probability distribution. The learning task for SRL models then consists of the two parts of learning the structure, and learning the parameters. In Markov logic networks, for example, the structure consists of all the logical clauses (13), and the parameters of the numeric weights attached to these clauses. In the ProbLog probabilistic logic programming language, the structure consists of clauses of the form (14), together with probabilities assigned to certain ground facts. In RBNs, the structure consists of the functional form of probability formulas, and parameters are the constants (29) of the formula. Since the structure represents interpretable, meaningful dependencies between different relations and attributes, it may also be elicited (at least in part) by domain experts.

▶ **Example 18.** Consider again the scenario of Example 3, and suppose one wants to create an SRL model for predicting whether a person should be labeled as an influencer. A social network expert would probably be able to say that whether or not a person is an influencer depends (maybe among other factors) on the number of his/her followers. In a probabilistic logic programming framework, this would lead us to include the clause

$$\mathit{influencer}(X) \leftarrow \mathit{follower}(Y,X) \tag{57}$$

in our model. The clauses in Markov logic networks do not represent directed implications like (57) but undirected logical properties or features that are deemed relevant for the probability of a possible world. Our knowledge about a connection between the *influencer* attribute and the *follower* relation can be incorporated into the model by constructing several features that express combinations of these two:

$$\begin{array}{l} \mathit{influencer}(X) \vee \mathit{follower}(Y,X) \\ \mathit{influencer}(X) \vee \neg\mathit{follower}(Y,X) \\ \neg\mathit{influencer}(X) \vee \mathit{follower}(Y,X) \\ \neg\mathit{influencer}(X) \vee \neg\mathit{follower}(Y,X) \end{array} \tag{58}$$

The fact that a greater number of followers increases the probability for being an influencer would here be encoded not already through the structure of the model, but by the relative magnitudes of the numeric weights associated with these four different features. Relational Bayesian networks again encode directed dependencies. Here our knowledge would imply that the probability formula for the *influencer* attribute should include the construct

$$\begin{array}{lll} F_{\mathit{influencer}(X)} \equiv & \ldots & \\ & & \texttt{COMBINE} < \mathit{probability\ formula} > \\ & & \texttt{WITH} < \mathit{combination\ function} > \\ & & \texttt{FORALL}\ Y \\ & & \texttt{WHERE}\ \mathit{follower}(Y,X) \\ & \ldots & \end{array} \tag{59}$$

This is only a partial specification of the probability formula for *influencer*, which leaves open the details of how the dependency on the number of followers should be aggregated in the combination function construct, and what other dependencies of the *influencer* attribute need to be encoded in its probability formula.

There is an apparent decrease in the ease-of-use from the logic programming via the Markov network to the RBN framework for encoding expert knowledge in the model structure. The first two frameworks allow modular specifications where different pieces of knowledge can be represented by separate logic-based representations. In the case of RBNs, all relevant knowledge for the attribute *influencer* needs to be collected in the single probability formula $F_{\mathit{influencer}(X)}$. If additional knowledge was provided that the *influencer* attribute also depends on a known *has_youtube_channel* attribute, then in a probabilistic logic or Markov logic framework this could be incorporated by adding new clauses, leaving the existing (57)(58) untouched. In RBNs, on the other hand, the additional knowledge regarding *has_youtube_channel* needs to be integrated with the previous knowledge inside the formula (59).

However, modular specifications, though intuitive on the surface, pose their own challenges. In pure logic, the total knowledge expressed by a set of formulas is simply the conjunction of the knowledge expressed by each single formula. In a logic-based, modular specification of a generative probabilistic model, the semantic contribution of each model component to the overall probability distribution defined by the model can not be defined by a "local semantics" of the component. The impact of each component on the probabilities defined always depends on the full model that it is part of. In short, syntactic modularity here does not translate into semantic modularity.

It should be apparent, now, that even though SRL frameworks support the integration of domain knowledge into the model construction process, this can not happen without a thorough understanding of the semantics and algorithmics of the SRL framework being used. Thus, expert-driven model development here requires both a domain and an SRL expert.

This example has highlighted the ability to (partially) construct the structure of an SRL model manually based on domain knowledge. It is, of course, a central objective to also use machine learning for determining the structure of a model, which leads to search and optimization problems in very large combinatorial spaces of possible model structures. For probabilistic logic programming frameworks, often search techniques from the field of inductive logic programming are adapted (e.g.[5]). Structure learning for Markov logic networks has received a particularly large amount of attention. Here inductive logic programming techniques have also been exploited [43, 50]. Other approaches (e.g. [38]) exploit more novel machine learning techniques, but still include an element of heuristic (beam) search over possible clauses. For RBNs in the structure learning problem has only been addressed in the context of a somewhat simplified framework [33].

A (graph) neural network model also consists of a structure (here often called the architecture) and its parameters. However, the balance between the tasks of structure design, or structure learning, on the one hand, and parameter learning on the other hand, is quite different. Whereas in SRL structure learning is perhaps the greatest and most fundamental challenge, one would typically view the learning problem of a GNN almost exclusively as a parameter optimization task. The network architecture may either be taken to be a given "standard solution", or obtained from a manageable set of candidate architectures via tuning such hyperparameters as the number and dimensions of network layers. The increasing complexity and variability of available neural components, however, also has given rise to the field of *neural architecture search* [14], which begins to share some characteristics with SRL structure learning.

12.2 Parameter learning

An SRL model usually contains only a moderate number of numerical parameters (the bound < 100 parameters probably covers a large fraction of SRL models presented in the literature). When these parameters have a clear, interpretable, statistical meaning then even parameters may be amenable to specification by domain experts. For example, in the probability formula

$$F_{\mathit{influencer}(X)} \equiv \texttt{WIF}\ \mathit{has\_youtube\_channel}(X)\ \texttt{THEN}\ 0.35\ \texttt{ELSE}\ 0.01$$

the parameters 0.35 and 0.01 correspond to the statistical frequencies of influencers among people who do, respectively do not, own a Youtube channel. In the absence of adequate training data, such parameters could be assessed (approximately) by human experts. In most cases, however, parameters of an SRL model should be learned from data.

Suppose, then, that an SRL structure has been fixed, and that this structure is parameterized by k real-valued parameters. Then a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^k$ defines an SRL model, i.e., for each domain V we have the probability distribution $P_V^{\boldsymbol{\theta}}$ on $\mathcal{G}(V, \mathcal{R})$ (the signature $\mathcal{R}$ always being fixed). Training data consists of a number of observed graphs $(V_1, \boldsymbol{R}_1), \ldots, (V_N, \boldsymbol{R}_N)$. The domains of the training graphs may be different, may be all the same $V_1 = \ldots, V_N$, or the data may only consist of a single graph ($N = 1$). In all cases, we can score $\boldsymbol{\theta}$ by the log-likelihood:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P_{V_i}^{\boldsymbol{\theta}}(\boldsymbol{R}_i). \tag{60}$$

In the case of Markov logic networks, the exact probabilities $P_{V_i}^{\boldsymbol{\theta}}(\boldsymbol{R}_i)$ are intractable to compute, and an approximate *pseudo-likelihood* is used instead. We note that our objective (60) does not contain a regularization term. This is because the problem of overfitting can arise (and must be dealt with) already at the stage of structure learning/design.

A major strength of probabilistic generative models lies in their ability to learn from incomplete data. Suppose our data consists of partially observed graphs $(V_1, \tilde{\boldsymbol{R}}_1), \ldots, (V_N, \tilde{\boldsymbol{R}}_N)$ where the domains V_i are fully observed, but for the relations we have only partial observations $\tilde{\boldsymbol{R}}_i$, meaning that for $R \in \mathcal{R}$ and entities $h, j \in V_i$ the atom $R(h, j)$ may have values *true*, *false* and *unknown*. Being a generative model, our current parameters then define for each possible completion $\boldsymbol{R}_i$ of $\tilde{\boldsymbol{R}}_i$ the probability

$$P_{V_i}^{\boldsymbol{\theta}}(\boldsymbol{R}_i | \tilde{\boldsymbol{R}}_i). \tag{61}$$

Taking the probability distribution over complete observations thus defined as an imputed complete dataset, one can then apply optimization techniques for complete datasets to optimize the parameter $\boldsymbol{\theta}$. Iterating the steps of imputing the *expected* complete data, and *maximizing* the likelihood function for the imputed data gives the famous *Expectation-Maximization (EM)* algorithm for learning from incomplete data. The EM algorithm is a general paradigm of almost universal applicability in statistical learning. However, in order to be feasible in practice for a particular model class (SRL or other), efficient techniques have to be developed for that particular model class to implement the computation of expected completions, and the subsequent optimization of the parameters. The expectation step often is computationally quite expensive, which means that learning from incomplete data usually is significantly more time consuming than learning from complete data.

GNN parameters are learned by minimizing a *loss function.* When the task for which a GNN is trained is classification, then usually the *cross-entropy loss* is used. Assume, for example, that the task is classification of a Boolean node label $C(X)$, and that we use a GNN architecture along the lines shown in Figure 2 on the left, where the output layer applies a *softmax* function to guarantee that the outputs represent a probability distribution over the possible class labels. Training data will consist of labeled nodes, which in general could be given by training examples

$$(V_1, C(j_1), \boldsymbol{R}_1), \ldots, (V_N, C(j_N), \boldsymbol{R}_N)$$

where j_i is a node in V_i, and $C(j_i) \in \{\mathit{true}, \mathit{false}\}$ is the observed label. Again, a fixed signature $\mathcal{R}$ for attributes and relations (other than the class label C) is given, and $\boldsymbol{R}_i$ consists of complete observations of $\mathcal{R}$ for V_i. Often, all examples will come from a single graph, i.e. $V_1 = \ldots = V_N$ and $\boldsymbol{R}_1 = \ldots, \boldsymbol{R}_N$. Let $\boldsymbol{\theta}$ be a setting for the weights in the network. Then the network produces outputs $\boldsymbol{o}^{\boldsymbol{\theta}}(j_i)$ that are 2-dimensional non-negative vectors for which $\boldsymbol{o}^{\boldsymbol{\theta}}(j_i)[0] + \boldsymbol{o}^{\boldsymbol{\theta}}(j_i)[1] = 1$. Assuming that the first output component is associated with the label *true*, and the second with the label *false*, we then obtain the cross-entropy loss

$$-\left(\sum_{i:C(j_i)=true} \log \boldsymbol{o}^{\boldsymbol{\theta}}(j_i)[0] + \sum_{i:C(j_i)=false} \log \boldsymbol{o}^{\boldsymbol{\theta}}(j_i)[1] \right). \tag{62}$$

Under the probabilistic interpretation of the outputs, and the assumption that all training examples are independent, this is the negative log-likelihood. In particular, considering the case $(V_i, \boldsymbol{R}_i) = (V_{i'}, \boldsymbol{R}_{i'})$, (62) incorporates the conditional independence assumption (26) for the distribution $P_V(C|\mathcal{R})$. Thus, the loss minimization objective of GNN training here is exactly the same as the likelihood maximization objective in learning an SRL model for the conditional distribution $P_V(C|\mathcal{R})$, if the SRL model makes assumption (26). The latter is the case, for example, when $P_V(C|\mathcal{R})$ is specified by an RBN consisting of a single probability formula $F_{C(X)}$. Parameter (weight) vectors $\boldsymbol{\theta}$ of GNNs are usually much larger than those of SRL models, and overfitting can also occur as a result of pure parameter learning. Therefore, the cross-entropy loss (62) will often be combined with a regularization term for $\boldsymbol{\theta}$.

12.3 From sparse to over-parameterizations

As noted in the preceding sections, SRL and GNN models are typically distinguished by huge differences in the size of their parameterizations. SRL models combine structure that encodes relevant features and dependencies with a sparse parameterization that quantifies these dependencies. GNNs follow the deep learning philosophy that feature discovery is automated as a part of the parameter learning problem [22, Chapter 1]. The use of high-dimensional parameter spaces in GNN architectures serves two distinct purposes:

- *Model capacity:* providing a rich hypothesis space that can capture complex relevant features.
- *Facilitating optimization:* gradient descent is more effective in higher-dimensional spaces.

We speak of over-parameterization when a model architecture contains more tunable parameters than are actually required to perfectly capture a target concept. As we have observed in Example 10, neural network training can be more effective in overparameterized than 'minimally sufficient' model architectures. This somewhat counter-intuitive observation has been made and studied by many authors, e.g. [52, 9]. A partial explanation is provided by consideration of the limit case where the dimensions of hidden layers (and number of weight parameters) goes to infinity [6, 2]. In this limiting case, the last hidden layer will contain sufficiently rich features (regardless of the weight settings at lower layers) such that learning can be reduced to a convex optimization problem for the weights at the output layer [6].

13 Conclusion and Outlook

We have studied similarities and differences in graph and network analysis using the tools of statistical relational learning and graph neural networks. We have emphasized the commonalities of these two paradigms, especially with regard to SRL frameworks of the Bayesian network constructor type. In particular for the relational Bayesian network framework we demonstrated the capability to directly encode GNNs without modifications or additions to the original RBN framework. This directly enables forms of neuro-symbolic integration by RBN models that combine neural encoding components with higher-level symbolic representations. As we have seen in Section 9, this can be exploited to tackle a larger variety of inference tasks, and to combine learning with expert-driven model specifications. However, in order to obtain maximal benefits from such combinations, several challenges have still to be met:

- Interpretability: symbolic representations are typically more interpretable for a human user than a neural network model. However, an RBN component that directly encodes a GNN module is not more interpretable than the original GNN. A challenge therefore is whether a learned RBN containing over-parameterized GNN components can be reduced to a smaller and more interpretable model, e.g. by some form of model distillation. In contrast to other approaches towards interpretability via surrogate model learning (e.g. [61]), the original and the simplified model here would live both in the same hypothesis space of RBNs.
- Trading structure learning for parameter learning: the success of GNN technology indicates that the overall strategy of reducing the learning problem as much as possible to a parameter learning problem has advantages over the SRL strategy that places primary importance on the structure of a model. This leads to the question of whether structure learning can be reduced to parameter learning. A very small-scale and simplistic instance of this was already presented in [35], where it was suggested to learn the appropriate combination function in a model by learning a mixture model over several candidate combination functions, and then selecting the one with the dominant coefficient in the learned mixture.
- Closing the efficiency gap: as discussed in Section 9, an RBN encoding of a GNN is mathematically equivalent, and, in principle, has the same parameter learning complexity as the original GNN. In practice, however, there is a significant gap, because the general purpose RBN algorithms do not automatically leverage the limited functional structures encountered in GNN encodings.

References

1 Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of IJCAI 2021*, 2021.

2 Francis Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017.

3 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

4 Pablo Barceló, Egor Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020.

5 Elena Bellodi and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, 15(2):169–212, 2015.

6 Yoshua Bengio, Nicolas Le Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. *Advances in neural information processing systems*, 18:123, 2006.

7 J. S. Breese, R. P. Goldman, and M. P. Wellman. Introduction to the special section on knowledge-based construction of probabilistic decision models. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(11), 1994.

8 Luitzen EJ Brouwer. Beweis der Invarianz des n-dimensionalen Gebiets. *Mathematische Annalen*, 71(3):305–313, 1911.

9 Alon Brutzkus and Amir Globerson. Why do larger models generalize better? a theoretical perspective via the xor problem. In *International Conference on Machine Learning*, pages 822–830. PMLR, 2019.

10 Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33, 2020.

11 Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. Scalable deep generative modeling for sparse graphs. In *International Conference on Machine Learning*, pages 2302–2312. PMLR, 2020.

12 L. De Raedt. *Logical and Relational Learning*. Springer, 2008.

13 R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1319–1325, 2005.

14 Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

15 Varun Embar, Sriram Srinivasan, and Lise Getoor. A comparison of statistical relational learning and graph neural networks for aggregate graph queries. *Machine Learning*, pages 1–20, 2021.

16 Herbert B Enderton. *A mathematical introduction to logic*. Elsevier, 2001.

17 Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

18 N. Friedman, Lise Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

19 Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pages 3419–3430. PMLR, 2020.

20 L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

21 Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

22 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: `http://www.deeplearningbook.org`.

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

24 William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1024–1034, 2017. URL: `http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs`.

25 D. Heckerman, C. Meek, and D. Koller. Probabilistic entity-relationship models, PRMs, and plate models. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning.* MIT Press, 2007.

26 Peter D Hoff, Adrian E Raftery, and Mark S Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.

27 Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

28 Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

29 Manfred Jaeger. Relational Bayesian networks. In Dan Geiger and Prakash Pundalik Shenoy, editors, *Proceedings of the 13th Conference of Uncertainty in Artificial Intelligence (UAI-13)*, pages 266–273, Providence, USA, 1997. Morgan Kaufmann.

30 Manfred Jaeger. On the complexity of inference about probabilistic relational models. *Artificial Intelligence*, 117:297–308, 2000.

31 Manfred Jaeger. Model-theoretic expressivity analysis. In L. De Raedt, K. Frasconi, P.and Kersting, and S.H. Muggleton, editors, *Probabilistic Inductive Logic Programming*, volume 4911 of *LNCS*, pages 325–339. Springer, 2008.

32 Manfred Jaeger. Probabilistic logic and relational models. In Reda Alhajj and Jon Rokne, editors, *Encyclopedia of Social Network Analysis and Mining*, pages 1–15. Springer New York, New York, NY, 2017. doi:10.1007/978-1-4614-7163-9_157-1.

33 Manfred Jaeger, Marco Lippi, Andrea Passerini, and Paolo Frasconi. Type extension trees for feature construction and learning in relational domains. *Artificial Intelligence*, 204:30–55, 2013. doi:10.1016/j.artint.2013.08.002.

34 Manfred Jaeger. Complex probabilistic modeling with recursive relational Bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 32:179–220, 2001.

35 Manfred Jaeger. Parameter learning for relational Bayesian networks. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

36 Jiuchuan Jiang and Manfred Jaeger. Numeric input relations for relational learning with applications to community structure analysis. *CoRR*, abs/1506.05055, 2015. arXiv:1506.05055.

37 K. Kersting and L. De Raedt. Towards combining inductive logic programming and Bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP-2001)*, Springer Lecture Notes in AI 2157, 2001.

38 Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude Shavlik. Learning Markov logic networks via functional gradient boosting. In *2011 IEEE 11th international conference on data mining*, pages 320–329. IEEE, 2011.

39 Angelika Kimmig, Bart Demoen, L De Raedt, V. Santos Costa, and Ricardo Rocha. On the implementation of the probabilistic logic programming language ProbLog. *Theory and Practice of Logic Programming*, 11(2-3):235–262, 2011. doi:10.1017/S1471068410000566.

40 Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*, 2014. arXiv:1412.6980.

41 Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint*, 2016. arXiv:1609.02907.

42 Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint*, 2016. arXiv:1611.07308.

43 Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448, 2005.

44 Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques.* MIT press, 2009.

45 Kathryn Blackmond Laskey. MEBN: A language for first-order Bayesian knowledge bases. *Artificial Intelligence*, 172(2-3):140–178, 2008. doi:10.1016/j.artint.2007.09.006.

46 Kathryn Blackmond Laskey and Suzanne M. Mahoney. Network fragments: Representing knowledge for constructing probabilistic models. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 334–341, San Francisco, CA, 1997. Morgan Kaufmann Publishers.

47 Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint*, 2018. `arXiv:1803.03324`.

48 Yao Ma, Suhang Wang, Chara C Aggarwal, Dawei Yin, and Jiliang Tang. Multi-dimensional graph convolutional networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 657–665. SIAM, 2019.

49 Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31:3749–3759, 2018.

50 Lilyana Mihalkova and Raymond J Mooney. Bottom-up learning of Markov logic network structure. In *Proceedings of the 24th international conference on Machine learning*, pages 625–632, 2007.

51 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4602–4609, 2019.

52 Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2018.

53 L. Ngo and P. Haddawy. Probabilistic logic programming and Bayesian networks. In *Algorithms, Concurrency and Knowledge (Proceedings ACSC95)*, Springer Lecture Notes in Computer Science 1023, pages 286–300, 1995.

54 Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

55 Giovanni Pellegrini, Alessandro Tibo, Paolo Frasconi, Andrea Passerini, and Manfred Jaeger. Learning aggregation functions. In *Proceedings of the Thirty International Joint Conference on Artificial Intelligence (IJCAI-21)*. International Joint Conferences on Artificial Intelligence, 2021.

56 Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

57 D. Poole. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

58 David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

59 Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph Markov neural networks. In *International conference on machine learning*, pages 5241–5250. PMLR, 2019.

60 Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 10(2):1–189, 2016.

61 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

62 M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

63 S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, third edition edition, 2010.

64 Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint*, 2020. `arXiv:2003.04078`.

65 Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.

66 T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)*, pages 715–729, 1995.

67 Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

68 Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

69 Gustav Šourek, Filip Železnỳ, and Ondřej Kuželka. Beyond graph neural networks with lifted relational neural networks. *Machine Learning*, pages 1–44, 2021.

70 J. van Mill. Domain invariance. Encyclopedia of Mathematics. URL: `http://encyclopediaofmath.org/index.php?title=Domain_invariance&oldid=16623`.

71 Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

72 Clément Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In *NeurIPS*, 2020.

73 Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pages 6487–6494. PMLR, 2019.

74 Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2021.

75 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

76 Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018.

77 Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: `https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf`.

78 Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

79 Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.