HetSAGE: Heterogenous Graph Neural Network for Relational Learning

Vince Jankovics, Michael Garcia Ortiz, Eduardo Alonso
City, University of London
{vince.jankovics, michael.garcia-ortiz, e.alonso}@city.ac.uk

Abstract

This paper aims to bridge this gap between neuro-symbolic learning (NSL) and graph neural networks (GNN) approaches and provide a comparative study. We argue that the natural evolution of NSL leads to GNNs, while the logic programming foundations of NSL can bring powerful tools to improve the way how information is represented and pre-processed for the GNN. In order to make this comparison, we propose HetSAGE, a GNN architecture that can efficiently deal with the resulting heterogeneous graphs that represent typical NSL learning problems. We show that on CORA, MUTA188 and MovieLens our approach outperforms the state-of-the-art in NSL.

1 Introduction

Many real world problems can be described by data that has inherent structure with distinct objects, properties and relationships. Learning and reasoning from data that has an inherent structure has always been a major interest in machine learning research. It has been showed that algorithms that can represent the structure of the data have a significant advantage over the ones that treat input features completely independent (De Raedt and Kersting 2008). During the early days of machine learning inductive logic programming (ILP) emerged as a framework to derive a generic theory that explains the observations while taking into account the constrains posed by the knowledge base.

Neuro-symbolic learning (NSL) emerged as a field to bridge the gap between symbolic (ILP) and neural network (NN) based approaches to be able to efficiently identify patterns in relational data (d'Avila Garcez et al. 2015). Propositionalisation based techniques rely on the transforming of a complex relational database into an attribute-value form that enables the use of NN models (Krogel et al. 2003). CILP++ (França, Zaverucha, and d'Avila Garcez 2014) is one of these that is considered state-of-the-art in the NSL community. However, due to the way how the model encodes the relational data, it has major limitations in terms of flexibility and scalability.

On the other hand, since NSL originates from logic based systems, logic programming provides a very powerful framework to express complex relationships and include additional information in a database, which could be beneficial for graph datasets as well. We provide a tool based on Prolog to transform a logic program to a graph with great flexibility.

Graph neural networks (GNN) has been an increasingly popular field in the last decade, with models that provide a way to identify patterns from graph structured data (Wu et al. 2019). Many relational databases (or logic programs) can be formulated as a knowledge graph or can be transformed into one without loss of information, which led to the realisation that the benefits of GNNs could be utilised in the context of relational learning. In this paper we argue that GNNs aim to solve the same problem as NSL systems, but they are more efficient as they fit the structure of the problem better, similarly to convolutional neural networks (CNN) are more suitable for images then multi-layer perceptrons (MLP).

In order to deal with graphs that originate from the relational learning domain, the GNN has to deal with heterogeneous nodes with edge features, which has been explored by (Zhang et al. 2019), but not in the context of relational learning. We developed a model (HetSAGE) to address these datasets efficiently and we provide experimental comparison between a popular ILP technique (Aleph), CILP++ and HetSAGE.

We also show that including additional connections that might not be part of the original database might benefit the GNN since it can provide richer flow of information between nodes. E.g. consider CORA (McCallum et al. 2000), which consists of papers with a directed edge between `p1` and `p2` if `p1` cites `p2`, but the inverse relationships (`p2` cited by `p1`) carries also relevant information, which is ignored by usual message passing GNN models that propage information along the edges.

Also, we include the target labels in the neighbouring nodes. The reasoning behind this choice is because it's typically important to know in an inductive learning setting what the neighbours of a datapoint would classify into, which is similar to the approach described in (Zhou et al. 2004). E.g. consider CORA again, if a new paper is presented to the model to predict which category it would fall into, then besides the bag-of-words or word2vec encoding of the content (Hamilton, Ying, and Leskovec 2017; Bojchevski and Günnemann 2018), including directly the labels of the neighbours as their features could be beneficial. However,

this requires additional care when propagating information between the nodes to avoid leaking information about the target node but still be able to utilise efficient batched training, which led to an extension of the neighbour sampling algorithm described in (Hamilton, Ying, and Leskovec 2017).

Our contributions are as follows: (i) we provide a comparison between CILP++ and HetSAGE to solve relational learning tasks, (ii) we provide a tool that transforms a logic program to a graph, (iii) we study the inclusion of the target labels in the neighbouring nodes, which is similar to the approach described in (Zhou et al. 2004). Our study also shows that the natural evolution of NSL leads to GNNs, providing a significant boost to the performance of state-of-the-art in NSL. Additionally, we show how logic programming can be use alongside GNNs to formulate and enrich the graph database for the GNN.

The source code of our model can be found is available online[1].

2 Related work

Neuro-symbolic learning Neural-symbolic systems provide an approach to deal with the probabilistic nature of data by integrating symbolic knowledge with connectionist approaches.

The CILP++ system (França, Zaverucha, and d'Avila Garcez 2014) is a neuro-symbolic learner that transforms first the relational learning problem to an attribute-value problem through propositionalization. For an overview on propositionalization techniques see (Krogel et al. 2003). CILP++ relies on the bottom clauses constructed by Aleph.

The generated propositional sentences are converted into a one-hot encoding as the input for the neural network. Each distinct clause in the body of the examples are tokenized and set to 0 if it is not present and 1 if it is. It's important to note that this encoding is the same as the bag-of-words approach from natural language processing (NLP). The only difference between the two is the grammar of the underlying sentence that the feature vector represent.

For example if a movie dataset had two examples, both directed by `director1`, but one with `actor1`, the other `actor2`, then it would turn into the feature vectors [1, 1, 0] and [1, 0, 1] with the first element representing `directed(director1)`, the second `has_actor(actor1)` and the third `has_actor(actor2)`.

The fact that the input vector encodes all the possible bottom clauses in a dataset creates limitations on the flexibility of the model. In terms of flexibility, consider that if a new movie comes out that has 2 directors that would not be possible to represent in this fixed encoding.

The scalability does not present a significant practical issue because of the increasing computational requirements, but it raises the issue that is commonly referred to as the "curse of dimensionality". The first description of the phenomena comes from (Bellman 1957) and it is been studied extensively in the context of machine learning (for more details see (Bishop 2007, Section 1.4)). For example in (França, Zaverucha, and d'Avila Garcez 2014) the MUTA188 dataset, that has 188 examples, has over a 1000 input features, which raises the question of how a model could generalize when there are more dimensions than data points in the input space.

The propositionalization and transformation to a numerical vector of the examples also strip away important information from the data. As an analogy, image classification reached decent performance when convolutional neural networks were introduced.

The main benefit of learning convolutional filters is to exploit the implicit bias in the images, i.e. that pixels close to each other form features that can be applied across the whole image (shift-invariance). On the contrary, previous methods that relied on vectors of the flattened image passed to MLP models did not have any of this structure preserved. A pixel in the top left corner of the image had the same relevance to the bottom left corner as a neighbouring pixel, meaning the pixels could be shuffled and the model would perform the same way (after retraining on the shuffled version), which does not happen with CNNs. The same applies to bag-of-worlds models, the sentences "A dog chasing a cat" and "A cat chasing a dog" is equivalent, while for models exploiting the inherent structure (i.e. the order of worlds) can make better sense of them.

ILP provides a way to learn rules from relational data where there are relationships between samples and features. The learning system finds solutions that obey the constraints posed by the background knowledge and the provided mode declarations, i.e. it is impossible to find a theory that has contradictions in it, like `has_car(A,B),long(B),short(B).` However, by turning the symbolic representation into a subsymbolic one these constraints are not respected by the model, i.e. soundness is not guaranteed. In simple terms, the reasoning process becomes a pattern recognition process where the model (neural network) identifies patterns in the clauses that correlate with the target label.

Graph neural networks Graph neural networks (GNN) emerged as a natural method to deal with graph data. For a thorough survey on the topic see (Wu et al. 2019).

The basic principle is similar to convolutional neural networks used for image processing, as 2D convolutions on an image is a special case of graph convolutions (Niepert, Ahmed, and Kutzkov 2016; Wu et al. 2019) where each neighbouring pixel is connected to the central one and the kernel size determines how far the filter aggregates the pixels or features. In the GNN case the connections can encode arbitrary relationships, e.g. bond type for molecular data or citations for publication databases.

Contrary to CILP++, GNNs preserve and exploit the structure of the data similarly to CNNs for images. Furthermore, GNNs can handle arbitrary input sizes since they rely on learned functions that are applied across the nodes of the graph, so for example having a paper with more citations is not a problem, while the propositionalization would pose a limitation.

[1] https://bit.ly/2FKzv05

Learning from large graphs has been challenging, since feeding the whole graph to a model might not be possible. In (Hamilton, Ying, and Leskovec 2017) they introduce a sampling technique that uniformly selects nodes from the target node's k-step neighbourhood and aggregates it to it. However, this architecture assumes just a single edge type, while our targeted datasets typically have several different relationships. In (Gilmer et al. 2017) they introduce a message passing architecture that takes into account edge types (or features).

In (Zhang et al. 2019) they explored learning from heterogeneous graphs, but not in the context of relational learning. In their work each type of information has its own neural network and an LSTM is used to combine the output of all, but we use a single neural network for each type of node since our data does not contain more complex information (image, time series, etc.) as node attributes.

3 Methodology

3.1 Knowledge representation

Graph Graphs are structures that are represented by a pair $G = (V, E)$ where V is a set of nodes (or vertices) and $E \subseteq V^2$ is a set of pairs of nodes (called edges), which are ordered when G is a directed graph, i.e. the pairs x, y and y, x are two distinct elements in E. Each node and edge can have associated labels or features.

In general, graphs can encode arbitrary information about the nodes and edges using $\mathbf{x}_v \in \mathbb{R}^c$ node and $\mathbf{x}_e \in \mathbb{R}^d$ edge feature vectors where c and d are the number of node and edge features, respectively.

There are homogeneous graphs that only contain nodes of the same type and heterogeneous graphs that contain different types of nodes. E.g. CORA is a homogeneous graph that represents a paper with a node and each edge represents that a paper cites another one, while MovieLens is a heterogeneous graph that represents relationships between movies, directors, actors and users. The main challenge in learning from heterogeneous graphs is due to the fact that different type of nodes can have different attributes, e.g. a movie has title, year, genre and rank, while an actor has name and gender.

Graphs can be used to represent databases similarly to logic programs (e.g. Prolog). Specifically, knowledge graphs are used to capture relational structures between entities. For example search engines (such as Google) use graphs to store knowledge gathered from the internet, or Wikidata captures information about articles on Wikipedia and relationships between them.

Logic programming Logic programming is a programming paradigm that is based on formal logic. Aleph and CILP++ relies on Prolog to handle the data and for our GNN we use the same framework to pre-process the data and transform the logic program to a graph. In this section we introduce the foundations of Prolog and describe the transformation algorithm.

In Prolog the domain of discourse D is represented as relations defined by clauses. Each clause is in the form of `head:-body` which translates to the logic formula $body \implies head$.

The head of the clause can only contain a single predicate while the body consists of a set of predicates (called goals) joined by conjunction and disjunction. A predicate is a compound term in the form of $r(a_1, a_2, \ldots, a_n)$ where r is a relationship and a_i is an atom, for example $human(john)$, $mother(mary, john)$, etc. A predicate can be represented as an ordered tuple in D. The number of arguments (or length of the tuple) n is referred to as arity.

Clauses can be split into two distinct groups, facts and rules. Facts are axioms that are assumed to be true and they are the special case where the body of the clause is empty (always true), i.e. `head:-true`, while rules are theorems that allow new inferences.

The main benefit of using rules comes with the use of variables as arguments which can stand for arbitrary terms. The variables that appear in the head are implicitly universally quantified while variables that appear only in the body are implicitly existentially quantified (Clocksin and Mellish 2003). For example this means that the rule `p2(X,Y):-p1(X,Z),p1(Z,Y)` translates to the first order logic statement:

$$\forall X, Y \exists Z, p1(X, Z) \wedge p1(Z, Y) \implies p2(X, Y) \quad (1)$$

Prolog relies on the closed world assumption, meaning a statement is only true if it is in the domain of discourse, and false otherwise, i.e. if it is a fact or it is provable through the rules. Given a query to prove, i.e. to assert if the statement is a consequence of D, Prolog uses backward-chaining to recursively prove the goals in the body of the statement. During the proof search all free variables are instantiated, i.e. replaced by constants, and the instantiations that prove the query are enumerated. For example the above example with facts `p1(a1,a2)`, `p1(a2,a3)`, `p1(a2,a4)`, `p1(a1,a5)` and query `p2(X,Y)` would return {`X=a1,Y=a3`} and {`X=a1,Y=a4`}. We utilise this mechanism to transform the logic program to its grounded form D_g that only contains variable free terms but still entails the same consequences. In practice this means that the same answers are returned for a given query in D and in D_g, i.e. $D \models S \iff D_g \models S$ where S is a statement.

Logic programs provide a way to represent information, which can be transformed to a graph without loss of information (and vice-versa), i.e. they are interchangeable (isomorphic) with a few constraints. A typical translation would map arity 0 terms (i.e. atoms, e.g. `a1,a2`) to nodes, arity 1 (e.g. `prop(a1)`) to node attributes and arity 2 (e.g. `rel(a1,a2)`) to labelled edges. Note, that properties can also be represented as relationships, e.g. the type of an entity (cat or dog) can be represented in 2 different ways, i.e. `cat(a1),dog(a2` versus `is_a(a1, cat),is_a(a2, dog)`. This is typically a design choice, that can depend on which representation matches better the given problem.

Following the grounding, the logic program D_g is transformed into a graph based on the principles highlighted above. The graph transformation consists of 3 steps:

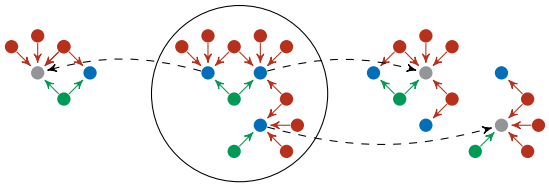

Figure 1: Our neighbourhood sampling: the initial graph in the circle with 3 node types (e.g. movie, actor, director) and blue nodes as targets. The 3 sampled neighbourhoods show the 2-step walk sampled around each target node. Target nodes in the sampled neighbourhood (gray) do not have the target label included (e.g. genre), while the surrounding movie nodes have them.

1. get all nodes and their types, e.g. in a movie database (MovieLens) node types are `movie`, `actor`, `director` and `user` with their corresponding unique identifier, such as `movie_1`, `actor_1`, etc.
2. get all node attributes, e.g. year for movies or gender for actors, etc.
3. get all relationships (edges), e.g. `director_1-directed-movie_1`, `user_1-rated_5-movie_1`, etc.

Relational database A relational database can be defined as a set of relations (referred to as tables) with each row representing a tuple. This shows the trivial connection between a grounded logic program and a relational database. A key difference is that arguments of a predicate in a logic program is identified by their position, while in a relational database they are identified with attribute names. This connection was utilised by (Motl and Schulte 2015) that led to the Relational Dataset Repository[2] that is being used in our study as the source of data.

3.2 HetSAGE

Neighbourhood sampling Our method uses a uniform sampling technique similar to GraphSAGE (Hamilton, Ying, and Leskovec 2017) with the difference that it creates non-overlapping subgraphs for each target node. This is done to enable efficient batching while including labels of the non-target neighbours. The sampled set of neighbours $\mathcal{N}_s(v)$ consists of n neighbours for each node v from $\mathcal{N}(v) = \{u|(v,u) \in E\}$. For a visual explanation see Figure 1.

Heterogeneous embedding The first step in our model is to embed each node's feature vector in a shared embedding space. The embedding $\mathbf{h}$ for node v is calculated according to:

$$\mathbf{h}^v = f_t^h(\mathbf{x}^v) \tag{2}$$

where f_e is the embedding neural network, t is the type of the node and $\mathbf{x}^v$ is the feature vector of node v. Note, that for each type t the learnable parameters of the neural network is shared for each node of type t.

[2]https://relational.fit.cvut.cz/

Message passing and readout layer After the embedding stage, the information between the nodes are propagated along the edges by the message passing module and a readout layer maps the hidden states of the target nodes to the predicted labels, similarly to (Gilmer et al. 2017). The graph convolutional layers aggregate the information from the surrounding of each node through N layers and aggregate the features from the last layer to produce a prediction y according to:

$$\mathbf{m}_{i+1}^v = \sum_{u \in \mathcal{N}_s(v)} f^m(\mathbf{e}^{vu})\mathbf{h}_i^u \tag{3}$$

$$\mathbf{h}_{i+1}^v = f_i^a(\mathbf{h}_i^v, \mathbf{m}_{i+1}^v) \tag{4}$$

$$y^v = f^r(\mathbf{h}_N^v) \tag{5}$$

where $\mathbf{m}_i^v$ is the sum of the messages passed to node v in layer i, and f^m, f_i^a, f^r are neural networks. Note, that the final node labels y^v are only calculated for $v \in V_t$, i.e. the set of target nodes from V.

We also included 1D batch normalisation applied to $\mathbf{h}_{i+1}^v$ after each message passing unit and also layer normalisation for $\mathbf{h}_i^v$ and $\mathbf{m}_{i+1}^v$ individually before they are passed to f_i^a. During the experiments we found this to be beneficial to balance the information carried forward from the node and the aggregated information from its neighbourhood.

4 Experiments

We conducted experiments on MovieLens, CORA and MUTA188 from the Relational Dataset Repository. The results are shown in Table 1. MovieLens is a heterogeneous graph learning problem with nodes representing `movies`, `actors`, `directors` and `users` and edges representing relationships such as `directed`, `rated`, etc. In CORA each node is a `publication` with the relationship `cited` between them. MUTA188 consists of 188 graphs, which are molecules with `atoms` and `bonds` between them and each graph has it's own label. In this work we represent the data as a single graph, with an additional node type `drug` that has the target labels assigned, and they are connected with the `has_atom` relationship to their corresponding atoms. This representation of the MUTA188 dataset matches how the other problems are defined, so it provides a proper comparison.

It is important to note that CILP++ performance on CORA was reported in (França, d'Avila Garcez, and Zaverucha 2015), but they don't mention how they turned the labels from multi-class to binary values. Our model is capable of dealing with the original problem with 7 classes, but CILP++ can only act as binary classifier (hence the N/A in the table). For comparison purposes we run the experiments with binary labels as well, assigning 1 to every paper that has the label *neural networks* and 0 otherwise (shown as CORA-binary). For the CILP++ experiments we used the architecture and hyperparameters from (França, Zaverucha, and d'Avila Garcez 2014). We present a comparison between three different versions of CORA, one with the bag-of-words encoding of the content of each paper and the paper labels, one without the content (*no words*) and one without

Dataset	CILP++ (%)	GNN (%)
MUTA188	89.74 (±5.32)	92.11 (±4.40)
CORA-binary no words	70.34 (±0.00)	93.68 (±2.15)
CORA-binary no labels	70.34 (±0.00)	92.92 (±0.98)
CORA-binary	69.70 (±0.07)	91.98 (±2.91)
CORA-multi no words	N/A	88.17 (±0.86)
CORA-multi no labels	N/A	81.85 (±0.58)
CORA-multi	N/A	83.38 (±1.44)
MovieLens	80.12 (±0.76)	80.22 (±0.80)

Table 1: Accuracies averaged over 10 random train/test split. HetSAGE outperforms CILP++ on each benchmark. CORA without the contents of the papers outperforms the experiments that include them, which could be because the propagated labels carry more information than the contents, and the model does not need to distinguish between the noise and the more relevant information. Including the labels seem to improve the performance. model.

the labels (*no labels*) to evaluate the benefit of label propagation.

For the CILP++ experiments we used the architecture and hyperparameters from (França, Zaverucha, and d'Avila Garcez 2014), our implementation can be found online[3]. We present a comparison between three different versions of CORA, one with the bag-of-words encoding of the content of each paper and the paper labels, one without the content (*no words*) and one without the labels (*no labels*) to evaluate the benefit of label propagation.

5 Conclusion

We demonstrated that HetSAGE is capable of handling problems defined as a logic program and that our model outperforms CILP++. We argue that GNNs in general are more suitable for addressing structured data than NSL. We additionally implemented a transformation that takes a logic program and generates a graph from it. This can also provide a powerful tool to include additional information (or common sense) in the data. Future work comprises conducting a larger scale comparison on what additional information helps the model and a more thorough hyperparameter and architecture search on HetSAGE.

References

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press. ISBN 978-0-691-07951-6.

Bishop, C. M. 2007. *Pattern Recognition and Machine Learning*. New York: Springer-Verlag New York Inc., newer edition. ISBN 978-0-387-31073-2.

Bojchevski, A.; and Günnemann, S. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. *arXiv:1707.03815 [cs, stat]* .

Clocksin, W. F.; and Mellish, C. S. 2003. *Programming in Prolog*. Berlin ; New York: Springer-Verlag, 5th ed edition. ISBN 978-3-540-00678-7.

d'Avila Garcez, A.; Besold, T. R.; de Raedt, L.; Földiak, P.; Hitzler, P.; Icard, T.; Kühnberger, K.-U.; Lamb, L. C.; Miikkulainen, R.; and Silver, D. L. 2015. Neural-Symbolic Learning and Reasoning: Contributions and Challenges. In *2015 AAAI Spring Symposium Series*.

De Raedt, L.; and Kersting, K. 2008. Probabilistic Inductive Logic Programming. In De Raedt, L.; Frasconi, P.; Kersting, K.; and Muggleton, S., eds., *Probabilistic Inductive Logic Programming: Theory and Applications*, Lecture Notes in Computer Science, 1–27. Berlin, Heidelberg: Springer. ISBN 978-3-540-78652-8.

França, M. V. M.; d'Avila Garcez, A. S.; and Zaverucha, G. 2015. Relational Knowledge Extraction from Neural Networks. In *CoCo@ NIPS*.

França, M. V. M.; Zaverucha, G.; and d'Avila Garcez, A. S. 2014. Fast Relational Learning Using Bottom Clause Propositionalization with Artificial Neural Networks. *Machine Learning* 94(1): 81–104. ISSN 1573-0565.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. *arXiv:1704.01212 [cs]* .

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*, 1024–1034.

Krogel, M.-A.; Rawles, S.; Železnỳ, F.; Flach, P. A.; Lavrač, N.; and Wrobel, S. 2003. Comparative Evaluation of Approaches to Propositionalization. In *International Conference on Inductive Logic Programming*, 197–214. Springer.

McCallum, A. K.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval* 3(2): 127–163.

Motl, J.; and Schulte, O. 2015. The CTU Prague Relational Learning Repository. *arXiv preprint arXiv:1511.03086* .

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning Convolutional Neural Networks for Graphs. *arXiv:1605.05273 [cs, stat]* .

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2019. A Comprehensive Survey on Graph Neural Networks. *arXiv:1901.00596 [cs, stat]* .

Zhang, C.; Song, D.; Huang, C.; Swami, A.; and Chawla, N. V. 2019. Heterogeneous Graph Neural Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, 793–803. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-4503-6201-6.

Zhou, D.; Bousquet, O.; Lal, T. N.; Weston, J.; and Schölkopf, B. 2004. Learning with Local and Global Consistency. In *Advances in Neural Information Processing Systems*, 321–328.

[3] https://bit.ly/3hJwLNT