Towards Invertible Semantic-Preserving Embeddings of Logical Formulae

Gaia Saveri[1,2,*], Luca Bortolussi[2]

[1]Department of Computer Science, University of Pisa, Italy
[2]Department of Mathematics and Geosciences, University of Trieste, Italy

Abstract

Logic is the main formal language to perform automated reasoning, and it is further a human-interpretable language, at least for small formulae. Learning and optimising logic requirements and rules has always been an important problem in Artificial Intelligence. State of the art Machine Learning (ML) approaches are mostly based on gradient descent optimisation in continuous spaces, while learning logic is framed in the discrete syntactic space of formulae. Using continuous optimisation to learn logic properties is a challenging problem, requiring to embed formulae in a continuous space in a meaningful way, i.e. preserving the semantics. Approaches like [1] are able to construct effective semantic-preserving embeddings via kernel methods (for linear temporal logic), but the map they define is not invertible. In this work we address this problem, learning how to invert such an embedding leveraging deep architectures based on the Graph Variational Autoencoder framework. We propose a novel model specifically designed for this setting, justifying our design choices through an extensive experimental evaluation. Reported results in the context of propositional logic are promising, and several challenges regarding learning invertible embeddings of formulae are highlighted and addressed.

Keywords

Semantic-preserving Embeddings, Graph Generative Models, Graph Variational Autoencoders

1. Introduction

Logic, in its many variants, is arguably one of the most prominent languages used to represent knowledge, specify requirements and explanations for complex systems, and reason in a human-comprehensible way [2]. On the other hand, Graph Neural Networks (GNN, [3]) have reached state-of-the-art in relational learning, providing meaningful inductive biases such as permutation invariance and sparsity awareness [4, 5]. Lately, much attention has been put on combining symbolic knowledge representation and neural computations as performed by GNNs, towards solving combinatorial and logical reasoning tasks [6, 7]. In this context, leveraging GNNs to learn real-valued representations of logic formulae would open the door to the use of gradient-based optimization techniques in the space of formulae, hence moving requirement mining from a discrete to a continuous search problem. A key desiderata of such a model would be that of semantic consistency, i.e. formulae which are semantically similar should be mapped to vectors

NeSy 2023, 17th International Workshop on Neural-Symbolic Learning and Reasoning, Certosa di Pontignano, Siena, Italy
*Corresponding author.
gaia.saveri@phd.unipi.it (G. Saveri); lbortolussi@units.it (L. Bortolussi)
© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3432/paper15.pdf

which are close in the latent space. Current state-of-the-art in this field, for Linear Temporal Logic, is based on kernel methods, hence it learns a non-invertible function of formulae to their embeddings [1]. We address this drawback proposing a model based on the Graph Variational Autoencoder framework (GVAE, [8, 9]), whose objective is to construct an invertible mapping from the discrete syntactic space of logic formulae to a continuous space where semantic similarity is preserved.

Our Contribution Moving from the rationale of GVAE models for Direct Acyclic Graphs (DAG, [10]), we formulate a model able to encode the syntactic tree of a logic formula into a continuous discriminative representation. Towards the goal of making these embeddings semantic-preserving, we propose a principled strategy to enrich the learned latent space with semantic information. We show results in the context of propositional logic, which - despite its grammatical simplicity - already poses challenges arising from the intrinsic symmetry of propositions and from the syntax-semantic interplay, which we address via a series of justified design choices. Experimental results are promising and should be taken as a proof-of-concept towards extending this methodology to richer logic formalisms.

Related Work Learning autoencoder models for DAGs was originally addressed via sequence models [11, 12, 13] , then casted into a graph-related task in [10, 14, 15]. A significant body of work in this field is dedicated to molecular generation [16, 17, 18]. Learning generative models with either syntactic or semantic constraints is instead tackled in [19, 20], respectively. Constructing a semantic similarity measure for logic formulae is instead in [1].

2. Background

As mentioned in Section 1, in this work we restrict the investigation to the setting of propositional logic. Denoting by $\{x_i\}_{i=1}^n$ a set of propositional variables, where each x_i is a binary variable that can evaluate either to true (1) or false (0), the set of propositional formulae (hereafter referred to also as propositions or requirements) is defined recursively by the syntax $\varphi := 1 \mid x_i \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$. Semantics is defined recursively following the usual meaning of Boolean operators (see Appendix A.1 for details).

A convenient way to represent propositions in this context is by means of their abstract syntax tree (AST), i.e. a tree representation of the syntactic structure of the formula where internal nodes are the Boolean operators and the leaves are the propositional variables. Since Boolean operators are at most binary, the AST arising from propositional formulae are binary trees. This representation opens the doors to the application of Graph Neural Network (GNN, [3]) models, i.e. deep learning models natively able to handle graph-structured inputs.

In this work we wish to learn both an encoder and a decoder for mapping inputs $\mathbf{x}$ to and from continuous vectors $\mathbf{z}$. The Variational Autorencoder (VAE, [21]) framework is designed to learn simultaneously two parametric functions: a probabilistic generation network (decoder) $p_\theta(\mathbf{x}|\mathbf{z})$ and an approximated posterior distribution (encoder) $q_\phi(\mathbf{z}|\mathbf{x})$, by maximizing the evidence lower bound:

$$\mathcal{L}(\phi, \theta; \mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL[q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] \tag{1}$$

where $KL[q(\cdot)||p(\cdot)]$ denotes the Kullback-Leibler divergence between $q(\cdot)$ and $p(\cdot)$ and $p(\mathbf{z})$ the prior distribution over latent variables $\mathbf{z}$.

The VAE model has been generalized to graph structured data by employing a GNN as encoder (and possibly decoder) model [8]. Deep generative models for graphs are partitioned into two classes (borrowing the terminology from [22, 23]), following the way in which nodes and edges of the graph are produced: *sequential generating* if nodes and edges are processed following a predefined order on them, one at a time; *one-shot generating* if the adjacency matrix of the graph is generated all at once.

If auxiliary information $\mathbf{y}$ (such as category labels or semantic context) about the input $\mathbf{x}$ should be incorporated into the learning process, frameworks like the Conditional Variational Autoencoder (CVAE, [24, 25]) allow to control the generative process by imposing a condition on both the encoder (in this context defined as $q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})$), the prior ($p_\psi(\mathbf{z}|\mathbf{y})$, which is parametric) and the decoder ($p_\theta(\mathbf{x}|\mathbf{y}, \mathbf{z})$). The network is trained by optimizing the following conditional marginal log-likelihood (notation is the same of Equation 1):

$$\mathcal{L}(\phi, \theta, \psi; \mathbf{x}, \mathbf{y}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{y},\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{y}, \mathbf{z})] - KL[q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})||p_\psi(\mathbf{z}|\mathbf{y})] \tag{2}$$

While for euclidean input formats the vector $\mathbf{y}$ is commonly concatenated to the input $\mathbf{x}$ (resp. the latent $\mathbf{z}$) before the computation of the encoding (resp. decoding) functions, in the context of graph generative models several possibilities exist: in cases in which there is additional information for all the nodes of the graph, it is concatenated to the initial node states [26]; when $\mathbf{y}$ is a property of the whole graph, it can be either concatenated to the latent vector $\mathbf{z}$ before the decoding process [27] or incorporated into the GNN decoding architecture itself [28].

2.1. Kernel-based Logic Embeddings

Learning embeddings of logic formulae preserving semantic similarity is addressed in [1] by means of the kernel trick. In more detail, the problem of finding a similarity measure between Signal Temporal Logic (STL) formulae is tackled by defining a suitable kernel function (and a corresponding learning algorithm for it) mapping formulae to a subspace of the continuous Hilbert space L^2, resulting in a high value for semantically similar formulae (and a low value otherwise; we refer to Appendix B for more details). Being the Boolean operators a subset of the operators of STL, we can restrict the definition of the kernel in [1] to the context of propositional logic. Considering propositions with at most n variables, the set $\mathcal{T}$ of the 2^n possible variable configurations and denoting $\varphi(\tau) = 1$ (resp. $\varphi(\tau) = -1$) if the formula φ evaluated on $\tau \in \mathcal{T}$ returns true (resp. false), then we formulate a *Boolean* kernel (by restricting the definition of Boolean *STL* kernel in [1]) between formulae φ, ψ as:

$$k(\varphi, \psi) = \int_{\tau\in\mathcal{T}} \varphi(\tau) \cdot \psi(\tau)\, d\tau \tag{3}$$

and compute it using Monte Carlo approximation putting a uniform probability measure over the space of configurations $\mathcal{T}$. Experiments confirm that the Boolean kernel of Equation 3 maps propositional formulae in a semantic-preserving continuous space, as reported in Appendix B, where we also show its correlation with Jaccard similarity coefficient among

propositional formulae. Moreover, reducing the dimensionality of the embedding space using Principal Component Analysis (PCA) shows that it is sensible to consider an embedding space of dimension much lower than the number of formulae used to construct the kernel of Equation 3. We defer to Appendix B for experimental results on dimensionality reduction.

3. Logic Embeddings by Graph Neural Networks

The main objective of this work is to investigate algorithms for computing invertible embeddings of propositional logic formulae, starting from their syntax and exploring methods for including semantic information, towards the goal of learning a latent space characterising the semantic of the logic. As every formula can be represented by its AST without any loss of information (see Section 2 and Appendix A), architectures based on the GNN framework provide a sensible inductive bias for our setting. Moreover, as we require to learn both an encoder and a decoder for formulae, we find it meaningful to deploy a Graph Variational Autoencoder (GVAE) model. Formally, given a propositional formula φ sampled according to the algorithm detailed in Appendix C.1, we design the input to our model to be the rooted tree arising from its AST, i.e. inputs are graphs $G = (V, E)$ with set of vertices V corresponding to operators and variables (taken with their multiplicity) appearing in φ and set of directed edges $E \subseteq V \times V$ such that $(v_i, v_j) \in E$ if v_j is a main subformula of v_i in φ. Although normal forms exists for propositional logic, we decided not to adopt any of them for mainly two reasons: firstly, rewriting formulae in normal form usually involves increasing the number of terms (namely every formula in n variables written either in conjunctive or disjunctive normal form can have up to 2^n terms) and this can yield to scalability issues when the formula is used as input for a GNN; secondly, in this work we are using propositional logic as a proof of concept for models that we wish to deploy for temporal logics, which in general do not admit simple normal forms.

Commonly, GNN models (and in particular those based on the Message Passing paradigm [29]) organize computations in a *synchronous* update scheme, i.e. all nodes update their state and exchange messages simultaneously. However, the nodes of the graphs we consider in this context have well-defined dependency relationships, hence following the approach adopted in [10], we process nodes sequentially, i.e. we establish a topological order on the nodes of the input graph (given by the depth-first traversal of the tree) and we perform computations following an *asynchronous* scheme, by updating a node state only when all of its predecessors' states have been updated. Possibly, this order can be reversed (i.e. we consider the input DAG with edges in the opposite direction, adding a virtual root for the reversed graph connected to all the leaves of the original tree) and messages can be also exchanged in a bottom up scheme, originating what we refer to as a *bidirectional* architecture. In a sense, when considering the bidirectional message passing scheme, the way in which computations are performed resembles the way in which one would evaluate the input formula itself.

3.1. Encoding

The encoder is the network responsible for mapping the discrete input graph G to a continuous latent representation $\mathbf{z}$, and in this case consists of a GNN with a (possibly bidirectional) asynchronous update scheme, as described earlier in this Section. In our model, the encoder is

formulated by adapting the Graph Attention Network (GAT, [30]) to the asynchronous message passing scheme of [10], we defer to Appendix A.2 for the mathematical details.

The initial nodes hidden states are the one-hot-encoding of their type (note that increasing the number of variables allowed increases the number of node types). Then, once the message phase terminates, all nodes states are computed, and we take as output of the encoder out_e the state of the node without any successor (referred to as *end node*; if no such node exists, a virtual one is added to the input graph). In case of bidirectional encoding, a concatenation of the state of the end nodes of both computation directions is taken. As approximate posterior $q(\cdot)$ of Equation 1 and 2 we consider a Gaussian distribution, whose mean and variance are obtained by feeding out_e to two MLPs.

3.2. Decoding

Given a real-valued latent vector $\mathbf{z}$, the goal of the decoder is to convert it to a discrete graph G representing the AST of a formula φ. The main requirement for the decoder is validity, i.e. it should produce syntactically correct formulae. Towards this objective we decided (ablations are discussed in Section D.1) to design a decoder which implements syntactic rules architecturally, i.e. that does not allow the generation of invalid formulae. This is integrated in a node-wise generation procedure (derived from that of [10]) which constructs one node v_i at a time and updates its state $\mathbf{h}_{v_i}$ as (denoting by $\mathcal{N}(v_i)$ the set of v_i's predecessors and by x_{v_i} the one-hot-encoding of its type):

$$\begin{aligned} \mathbf{h}_{v_i} &= GRU(x_{v_i}, \mathbf{h}^{\text{agg}}_{v_i}) \\ \mathbf{h}^{\text{agg}}_{v_i} &= \sum_{v_j \in \mathcal{N}(v_i)} g(\mathbf{h}_{v_j}) \odot m(\mathbf{h}_{v_j}) \end{aligned} \tag{4}$$

where GRU is a Gated Recurrent Unit [31] and $\odot$ represents the gated sum between a mapping network m and a gating network g, implemented as MLPs.

Decoding thus results in the following top-down iterative algorithm:

1. Given the initial latent vector $\mathbf{z}$, the starting graph state $\mathbf{h}_G = \mathbf{h}_0$ is produced by feeding it to a MLP with softmax activation. By construction, the type of the first created node v_0 is *start*, i.e. it is a synthetic starting node;
2. Each node v_i, with $1 \leq i \leq \max_v$ (where $\max_v$ is an input parameter specifying the maximum number of nodes admitted during generation), is generated as:
   a) the type of v_i is inferred by first computing a distribution over all possible types using an MLP with softmax activation taking as input the current graph state $\mathbf{h}_G$, then sampling its type from such distribution;
   b) the hidden state of v_i is updated using Equation 4;
   c) the graph state is updated as: $\mathbf{h}_G = \mathbf{h}_{v_i}$;
3. Nodes are expanded following a depth-first traversal of the tree, taking into account the arity of the operators represented by node types, this prevents the violation of syntactic constraints;
4. The graph generation stops when the tree cannot grow further (i.e. all leaf nodes are variables), or when $i = \max_v$ (this latter case is the only possibility of having invalid formulae as output).

In Section 2.1 (and more in detail in Appendix B) we describe an effective methodology to compute finite-dimensional semantic embeddings of propositional formulae. We remark that, in order to construct such representations for a formula φ, it is not necessary to explicitly know φ itself, but only its valuations of a set of assignments $\{\tau_i\}_{i=1}^n \subseteq \mathcal{T}$. This makes it sensible to formulate a conditional variant of our model, i.e. a CVAE architecture using as semantic context vector the kernel embedding $\mathbf{y}_\varphi$ of the input formula φ (obtained after performing PCA dimensionality reduction), concatenated to the latent representation $\mathbf{z}$ before starting the decoding computations.

3.3. Learning

Training is performed using teacher forcing, i.e. the decoder is fed with the ground truth graph it has to reconstruct during loss computation. At each iteration of the decoding algorithm of Section 3.2, we accumulate the negative log-likelihood by feeding the network with a vertex having the ground truth node type, so that the network is forced to stay close to true node sequences.

4. Experiments

Experiments on a family of variations of our model pursue mainly two goals: measure VAE reconstruction and generative abilities and qualitatively evaluate the smoothness of the latent space. Hence, following [10, 32], we test:

- Abilities of VAE models: we conduct standard experiments to check reconstruction accuracy, prior validity, uniqueness and novelty;
- Abilities of CVAE models: apart from reconstruction accuracy experiments as those of the previous point, we test the capability of our model of preserving semantic similarity in the latent space;
- Latent space visualization: we qualitatively evaluate the ability of the latent space to capture characteristic structural features by visualizing it and interpolating it.

Code to reproduce the results is available at https://github.com/GaiaSaveri/LogicVAE.

4.1. Experimental Setting

We propose the following versions of our model (hereafter referred to as LogicGVAE):

- Syntax-only: given an input formula φ, it encodes its AST without any additional information, i.e. it has access only to the syntactic information of φ;
- Semantic-conditioned: given an input formula φ, we evaluate its semantic-context vector using the kernel of Section 2.1 and perform conditional decoding as described at the end of Section 3.2;

For both variants, we perform experiments on a set of different asynchronous encoding GNNs, namely GRU, Graph Convolutional Networks (GCN, [33]) and GAT. Architectural and training details of all models are specified in Appendix D.

Table 1
Results of VAE and CVAE abilities tests, percentages are averaged over 300 test formulae with 5 variables. The second line of the accuracy column for the GAT and GCN models represents the accuracy computed by considering only the most frequently decoded formula for each datapoint.

(a) Results of VAE abilities tests.

Encoder	Acc.	Val.	Uniq.	Nov.
GAT	**93.92**	89.38	**56.53**	**55.34**
	95.42			
GCN	91.27	**98.78**	29.91	28.32
	93.56			
GRU	82.24	75.52	10.63	19.16

(b) Results of CVAE abilities tests.

Encoder	Acc.	Val.	Sem. Dist.	Ker. Value
GAT	**87.43**	**93.72**	**6.317**	**0.7985**
	92.45			
GCN	85.35	89.14	6.808	0.6924
	91.79			

4.2. Experimental Results

4.2.1. VAE and CVAE abilities

This suite of experiments aims at evaluating: (a) accuracy, i.e. ability to perfectly reconstruct the input AST; (b) validity, i.e. ability to generate syntactically valid formulae; (c) uniqueness, i.e. proportion of distinct graphs out of valid generations; (d) novelty, i.e. proportion of generated graphs which are not in the training set. To evaluate accuracy, we encode each test graph G to get mean and variance of the posterior approximation $q_\phi(\mathbf{z}|G)$, then we sample 10 latent vectors from such distribution and decode each 10 times. Accuracy is defined as the percentage of these 100 graphs which are identical to the input G. Validity is instead computed as the proportion of syntactically valid AST obtained by sampling 1000 latent vectors from the prior $p(\mathbf{z})$ and decoding each 10 times. For what concerns the conditional variant of our architecture, $q_\phi(\cdot)$ and $p_\psi(\cdot)$ are those of Equation 2.

In the conditional setting, we test accuracy as described above, then we test how well the learned latent space preserves semantic-similarity by computing: (a) mean distance from the input context vector, i.e. given a semantic vector $\mathbf{y}$ on which we condition the decoding, we sample 100 vectors from the prior $\mathbf{z} \sim p_\psi(\mathbf{z}|\mathbf{y})$ and decode each 10 times, the most commonly decoded formula is kept for each $\mathbf{z}$ and its semantic embedding computed and compared with ground truth $\mathbf{y}$ using L_2 norm; (b) mean kernel similarity among formulae with same semantic vector $\mathbf{y}$, i.e. following the procedure of the previous point, we compute the kernel among the most decoded formulae for each $\mathbf{y}$.

Table 1a reports results of the tests checking VAE abilities for different encoders (described in full details in Appendix D). The first aspect to notice is that a convolution-style GNN (either GCN or GAT) outperforms a recursive one (GRU) on all performance indices. This might be due to the fact that convolutions exploit more local substructure patterns and they leverage the hidden state of the node we are updating, instead of only those of its neighbors, differently from a GRU architecture. The reconstruction accuracy of both GCN and GAT is high and comparable, however when encoding with a GAT we record a higher ability of the latent space to capture structural features of data, as witnessed by a higher percentage of unique and novel decoded formulae. These results can be leveraged in a requirement optimization scenario, by mapping a given formula in its latent representation and optimize its structure in the continuous

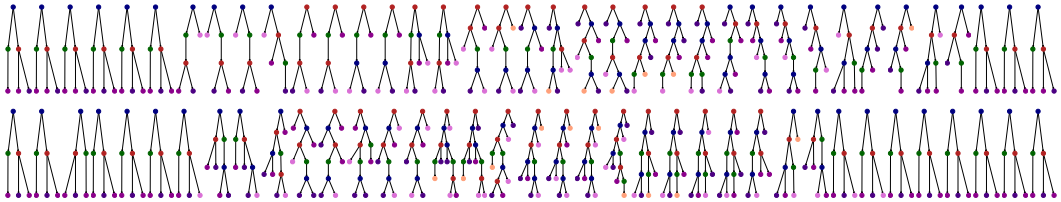

Figure 1: Slerp interpolation around a random formula of LogicVAE model with GAT (upper) and GCN (lower) encoder. Color of nodes encode their type.

embedding space.

Table 1b reports results of CVAE abilities tests (using as encoding networks only the most promising ones according to Table 1a). If we take a pool of 5000 random Boolean formulae with 5 variables, which we use to generate the semantic context vectors for each test instance, they have an average distance between semantic embeddings of 18.7325 and an average kernel similarity of 0.1695. Since the semantic distances and the kernel values reported in Table 1b are much lower (resp. higher), we observe that conditioning on a semantic vector our input data actually maps them in a space where semantic similarity is to some extent preserved.

It is worth noting that, although the reconstruction accuracy of the conditional architecture (Table 1b) is lower than that of the unconditional architecture (Table 1a), the former might be leveraged in a requirement-mining scenario, where only valuations of a formula in a set of configurations are observed (thus allowing the construction of the semantic context vector) and an explicit formula optimising a fitness function has to be found (i.e. decoded). Essentially, in this case we need to use the CVAE as a generative model, to sample formulae which (approximately) have the semantics specified by a given semantic context vector, obtained by solving an optimization problem in the semantic embedding space. This task cannot even be defined in a vanilla VAE scenario, where only syntactic information is exploited. Moreover, the reconstruction accuracy increases significantly if we compute it considering only the most frequently decoded formula for each test point, as reported as second row of the accuracy column of Table 1b.

More experimental results and ablation studies are reported in Appendix D, we underline in particular the cruciality of the syntactic constraints in the decoder architecture (Section 3.2).

4.2.2. Latent Space Visualization

We use spherical interpolation (*slerp*, [34]) around a given latent point to qualitatively evaluate the smoothness of the latent space. This briefly consists in interpolating points following a great circle of a hypersphere in a space having the same dimension of the latent space. Starting from a point $\mathbf{z}_\varphi$ encoding a formula φ, we follow a great sphere starting from $\mathbf{z}_\varphi$ and pick 35 evenly spaced points to decode. From Figure 1 we notice that the model using a GAT encoder produces a smoother latent space, as it changes less nodes at each step of the interpolation.

5. Conclusions and Future Work

In this work we propose LogicVAE, a configurable GVAE model built to learn a data-driven representation for propositional logic formulae. By leveraging an asynchronous message passing computational scheme in the encoder and ad-hoc syntactic constraints in the decoder, it is able to learn an expressive latent space. We provide a novel way to incorporate semantic information in the learning process, towards the goal of building an embedding space which is semantic-preserving. The main bottleneck of this approach is scalability to datasets with more than 5 variables, a possible solution is a hierarchical approach, sketched in Appendix D.2.

As already mentioned, results reported in this paper should be taken as a proof of concept towards extending the model to more complex formalisms, such as temporal logic, which come with the additional challenge of learning temporal and/or threshold parameters associated to operator (resp. variable) nodes, but which generally operates on few signals only (typically no more than 5 [35]). Moreover, inspired by AlphaCode [36], which defines a generative model for programs (another context in which the syntactic-semantic interplay has a fundamental role), we plan to both incorporate a self-supervised pre-training stage in our encoder (adapting the masked language modelling loss used in AlphaCode), and to check the viability of using large language transformer-based models [37] to encode and decode logic formulae.

Acknowledgments

This study was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043). This manuscript reflects only the Authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

[1] L. Bortolussi, G. M. Gallo, J. Kretínský, L. Nenzi, Learning model checking and the kernel trick for signal temporal logic on stochastic processes, in: Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I, volume 13243 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 281–300. doi:10.1007/978-3-030-99524-9_15.

[2] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach (4th Edition), Pearson, 2020. URL: http://aima.cs.berkeley.edu/.

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Networks 20 (2009) 61–80. doi:10.1109/TNN.2008.2005605.

[4] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A comprehensive survey on graph neural networks, IEEE Trans. Neural Networks Learn. Syst. 32 (2021) 4–24.

[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks, CoRR abs/1806.01261 (2018). URL: http://arxiv.org/abs/1806.01261. arXiv:1806.01261.

[6] L. C. Lamb, A. S. d'Avila Garcez, M. Gori, M. O. R. Prates, P. H. C. Avelar, M. Y. Vardi, Graph neural networks meet neural-symbolic computing: A survey and perspective, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, 2020, pp. 4877–4884.

[7] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: A methodological tour d'horizon, Eur. J. Oper. Res. 290 (2021) 405–421.

[8] T. N. Kipf, M. Welling, Variational graph auto-encoders, CoRR abs/1611.07308 (2016). URL: http://arxiv.org/abs/1611.07308. arXiv:1611.07308.

[9] X. Guo, L. Zhao, A systematic survey on deep generative models for graph generation, CoRR abs/2007.06686 (2020).

[10] M. Zhang, S. Jiang, Z. Cui, R. Garnett, Y. Chen, D-VAE: A variational autoencoder for directed acyclic graphs, in: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 1586–1598.

[11] K. S. Tai, R. Socher, C. D. Manning, Improved semantic representations from tree-structured long short-term memory networks, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers, The Association for Computer Linguistics, 2015, pp. 1556–1566.

[12] B. Shuai, Z. Zuo, B. Wang, G. Wang, Dag-recurrent neural networks for scene labeling, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, 2016, pp. 3620–3629.

[13] L. Mou, G. Li, L. Zhang, T. Wang, Z. Jin, Convolutional neural networks over tree structures for programming language processing, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA, AAAI Press, 2016, pp. 1287–1293.

[14] V. Thost, J. Chen, Directed acyclic graph neural networks, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, 2021.

[15] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, P. W. Battaglia, Learning deep generative models of graphs, CoRR abs/1803.03324 (2018).

[16] W. Jin, R. Barzilay, T. S. Jaakkola, Junction tree variational autoencoder for molecular graph generation, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 2328–2337.

[17] Q. Liu, M. Allamanis, M. Brockschmidt, A. L. Gaunt, Constrained graph variational autoencoders for molecule design, in: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018,

December 3-8, 2018, Montréal, Canada, 2018, pp. 7806–7815.

[18] V. Kondratyev, M. Dryzhakov, T. Gimadiev, D. Slutskiy, Generative model based on junction tree variational autoencoder for HOMO value prediction and molecular optimization, J. Cheminformatics 15 (2023) 11.

[19] M. J. Kusner, B. Paige, J. M. Hernández-Lobato, Grammar variational autoencoder, in: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of *Proceedings of Machine Learning Research*, PMLR, 2017, pp. 1945–1954.

[20] T. Ma, J. Chen, C. Xiao, Constrained generation of semantically valid graphs via regularizing variational autoencoders, in: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018, pp. 7113–7124.

[21] D. P. Kingma, M. Welling, Auto-encoding variational bayes, in: 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014.

[22] Y. Zhu, Y. Du, Y. Wang, Y. Xu, J. Zhang, Q. Liu, S. Wu, A survey on deep graph generation: Methods and applications, in: Learning on Graphs Conference, LoG 2022, 9-12 December 2022, Virtual Event, volume 198 of *Proceedings of Machine Learning Research*, PMLR, 2022, p. 47.

[23] X. Guo, L. Zhao, A systematic survey on deep generative models for graph generation, CoRR abs/2007.06686 (2020). URL: https://arxiv.org/abs/2007.06686. arXiv:2007.06686.

[24] K. Sohn, H. Lee, X. Yan, Learning structured output representation using deep conditional generative models, in: Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, 2015, pp. 3483–3491.

[25] O. Ivanov, M. Figurnov, D. P. Vetrov, Variational autoencoder with arbitrary conditioning, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, 2019.

[26] C. Yang, P. Zhuang, W. Shi, A. Luu, P. Li, Conditional structure generation through graph variational generative adversarial nets, in: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 1338–1349.

[27] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, P. W. Battaglia, Learning deep generative models of graphs, CoRR abs/1803.03324 (2018). URL: http://arxiv.org/abs/1803.03324. arXiv:1803.03324.

[28] Y. Li, L. Zhang, Z. Liu, Multi-objective de novo drug design with conditional graph generative model, Journal of Cheminformatics 10 (2018) 33.

[29] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of *Proceedings of Machine Learning Research*, PMLR, 2017, pp. 1263–1272.

[30] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[31] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, ACL, 2014, pp. 1724–1734.

[32] M. J. Kusner, B. Paige, J. M. Hernández-Lobato, Grammar variational autoencoder, in: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of *Proceedings of Machine Learning Research*, 2017, pp. 1945–1954.

[33] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[34] T. White, Sampling generative networks: Notes on a few effective techniques, CoRR abs/1609.04468 (2016).

[35] G. Ernst, P. Arcaini, I. Bennani, A. Donze, G. Fainekos, G. Frehse, L. Mathesen, C. Menghi, G. Pedrielli, M. Pouzet, S. Yaghoubi, Y. Yamagata, Z. Zhang, Arch-comp 2020 category report: Falsification, in: ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20), volume 74 of *EPiC Series in Computing*, 2020, pp. 140–152.

[36] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, O. Vinyals, Competition-level code generation with alphacode, Science 378 (2022) 1092–1097.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 5998–6008.

[38] O. Maler, D. Nickovic, Monitoring temporal properties of continuous signals, in: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings, volume 3253 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 152–166.

[39] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch, in: NIPS 2017 Workshop on Autodiff, 2017.

[40] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, A. Lerchner, beta-vae: Learning basic visual concepts with a constrained variational framework, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[41] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[42] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2019, pp. 2623–2631.

A. Extended Background

A.1. Propositional Logic

Propositional logic is a formal language whose grammar recursively defines well-formed formulae by combining variables $\{x_i\}_{i=1}^n$ and logical operators $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction) . More precisely, variables are valid formulae (also called atomic sentences), and given valid sentences α and β it holds that $\neg\alpha$, $\alpha \wedge \beta$ and $\alpha \vee \beta$ are valid sentences as well.

Semantics of propositional logic is defined starting from assignments τ of truth values to each propositional variable x_i then recursively applying the following rules to each sub-formula α (the valuation of α on τ is denoted by $\alpha(\tau)$): $\neg\alpha(\tau) = 1$ if $\alpha(\tau) = -1$, $(\alpha \wedge \beta)(\tau) = 1$ if both $\alpha(\tau) = 1$ and $\beta(\tau) = 1$, $(\alpha \vee \beta)(\tau) = 1$ if either $\alpha(\tau) = 1$ or $\beta(\tau) = 1$.

As mentioned in Section 2, every propositional formula φ can be represented by its Abstract Syntax Tree (AST), as showed in Figure 2.

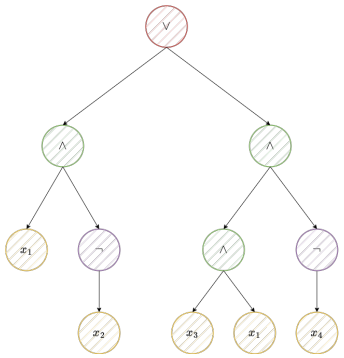

Figure 2: AST of the formula $(x1 \wedge \neg x_2) \vee ((x_3 \wedge x_1) \wedge \neg x_4)$. Color of the nodes encodes their type.

A.2. Graph Neural Networks

Graph Neural Networks (GNNs, [3]) are deep learning architectures that operate in the graph domain, i.e. they are built to solve graph related tasks in an end-to-end manner.

Arguably the most common framework for GNN is the so called Message Passing Neural Network framework (MPNN, [29]), which abstracts many commonalities between existing GNN models. As the name suggests, the defining characteristic of MPNNs is that they use a form of neural message passing in which real-valued vector messages are exchanged between nodes (in particular between 1-hop neighboring vertices), for a fixed number of iterations.

In detail, during each message passing iteration t in a MPNN, the hidden representation $\mathbf{h}_v^{(t)}$ of each node v is updated according to the information $\mathbf{m}_v^{(t)}$ aggregated from its neighborhood $\mathcal{N}(v)$ as:

$$\mathbf{h}_v^{(t+1)} = U_t(\mathbf{h}_v^{(t)}, \sum_{w \in \mathcal{N}(v)} M_t(\mathbf{h}_v^{(t)}, \mathbf{h}_w^{(t)})) \tag{5}$$

where M_t is called message function and U_t is called message update function: both are arbitrary differentiable functions (typically implemented as neural networks).

After running T iterations of this message passing procedure, a readout phase is executed to compute the final embedding $\mathbf{y}_v$ of each node v as:

$$\mathbf{y}_v = R(\{\mathbf{h}_v^{(T)} | v \in G\}) \tag{6}$$

where R (the readout function) needs again to be a differentiable function, invariant to permutations of the node states.

Hence, the intuition behind MPNNs is that at each iteration every node aggregates information from its immediate graph neighbors, so as iterations progress each node embedding contains information from further reaches of the graph.

A way to instantiate the message and update functions of Equation 5 is the Graph Convolutional Network (GCN, [33]) model, which generalizes the convolution operation to graph structured data by updating the representation $\mathbf{h}_v$ of each node v as:

$$\mathbf{h}_v^{(t+1)} = \sigma\left(\sum_{w \in \mathcal{N}(v)} \frac{1}{c_{vw}} h_w^{(t)} W^{(t+1)} \right) \tag{7}$$

with σ non-linear activation function and $c_{vw} = \{D^{\frac{1}{2}}(A + I)D^{\frac{1}{2}}\}_{vw}$ being A and D the adjacency and node-degree matrix of the input graph, respectively, and I the identity matrix.

Another possible computation scheme for the operations of Equation 5 is the Graph Attention Network (GAT , [30]) which uses self-attention to learn relative weights between neighboring nodes. It consists in:

$$\begin{aligned} \mathbf{h}_v^{(t+1)} &= \sigma(\sum_{w \in \mathcal{N}(v) \cup v} \alpha_{vw}^{(t+1)} W^{(t+1)} \mathbf{h}_w^{(t)}) \\ \alpha_{vw}^{(t+1)} &= \text{softmax}(\text{LeakyReLU}(\mathbf{a}^T [W^{(t+1)} \mathbf{h}_v^{(t)} || W^{(t+1)} \mathbf{h}_w^{(t)}])) \end{aligned} \tag{8}$$

being σ a non-linear activation function, $\mathbf{a}$ a vector of learnable parameters and $||$ a node-wise operation like concatenation or sum. Possibly GAT uses a multi-head attention mechanism to increase the model capabilities, by replicating the operations of Equation 8 multiple times independently and aggregating the results.

In its original version, the MPNN model is expected to exchange messages between nodes synchronously, however when the input graph is directed it is sensible to follow a partial order among nodes, as observed in [10]. In this *asynchronous* message passing scheme, each iteration t of Equation 5 is performed following the topological order established for the nodes of the input graph: in this scheme a node v must wait for all of its predecessors' updates before computing

its new hidden state $\mathbf{h}_v^{(t)}$, i.e. it uses the current layer representation of its direct neighbors $\{\mathbf{h}_w^{(t)}\}_{w \in \mathcal{N}(w)}$ before updating its state, instead of the one of the previous layer.

B. Kernel Trick for Logic Formulae

A kernel function for Signal Temporal Logic The kernel we start from when defining a semantic similarity measure for logic formulae is defined in [1]. It is originally developed for Signal Temporal Logic (STL [38]), which is a linear-time temporal logic suitable to express properties over real-valued trajectories. A trajectory in this context is a function $\xi : I \to D$ where $I \subseteq \mathbb{R}_{\geq 0}$ is a time domain and $D \subseteq \mathbb{R}^n, n \in \mathbb{N}$ is a state space (we also denote by $\mathcal{T}$ the space of trajectories). A key characteristic of STL is that it can be given both a *qualitative* (or Boolean) and a *quantitative* notion of satisfaction (the latter called robustness). More in detail, given a STL formula φ, a trajectory ξ and a time-point t, for the qualitative satisfaction we denote by $s(\varphi, \xi, t) = 1$ (resp. $s(\varphi, \xi, t) = -1$) if ξ at time t satisfies (resp. unsatisfies) φ, while for the quantitative satisfaction we denote by $\rho(\varphi, \xi, t) \in \mathbb{R}$ the robustness of φ for trajectory ξ, i.e. how robust is the satisfaction of φ w.r.t. perturbations in the signal. It holds that $\rho(\varphi, \xi, t) > 0 \to s(\varphi, \xi, t) = 1$ and $\rho(\varphi, \xi, t) < 0 \to s(\varphi, \xi, t) = -1$ (completeness property).

The definition of quantitative satisfaction allows to consider STL predicates φ as functionals mapping trajectories to their robustness, i.e. $\rho(\varphi, \cdot) : \mathcal{T} \to \mathbb{R}$, hence a formula can be embedded into a (possibly infinite-dimensional) Hilbert space. In this space we can consider as 'scalar product' between pairs of formulae φ, ψ the following:

$$k(\varphi, \psi) = \int_{\xi \in \mathcal{T}} \rho(\varphi, \xi) \cdot \rho(\psi, \xi) \, d\mu_0 \xi \tag{9}$$

being μ_0 a probability measure over trajectories. It can be proved that $k(\cdot, \cdot)$ of Equation 9 is a proper kernel function, and Probably Approximate Correct (PAC) bounds can be provided to give probabilistic bounds on the error committed when using the kernel for learning in formulae space. Experiments carried out in [1] confirms that the kernel of Equation 9 efficiently captures semantic similarity between STL requirements.

A kernel function for propositional logic Building upon the idea of the STL kernel defined above, we define a similarity measure between propositional formulae as detailed in Section 2.1.

The Jaccard index (or similarity coefficient) measures the similarity between two sets A and B as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$; given two propositions φ, ϕ and the sets of their assignment valuations X_φ, X_ψ resp. over variable configurations $\mathcal{T}$, the Jaccard index $J(\varphi, \psi) = \frac{|X_\varphi \cap X_\psi|}{|X_\varphi \cup X_\psi|}$ denotes the proportion of configurations in which the two formulae agree, hence measuring semantic similarity between its inputs. On the other hand, the value of $k(\varphi, \psi)$ represents the normalized difference between the set of configurations $\tau \in \mathcal{T}$ on which φ and ψ agree and disagree, being 1 in the case of semantically equivalent formulae. This quantity is positively correlated to the Jaccard index as $k(\varphi, \psi) = 2J(\varphi, \psi) - 1$, corroborating the claim that Equation 3 can be used as a measure of semantic similarity between formulae. Putting all together, we get that the kernel is an interpretable measure of similarity between formulae, for example a kernel of 0.8 can be understood as formulae agreeing on 90% of configurations.

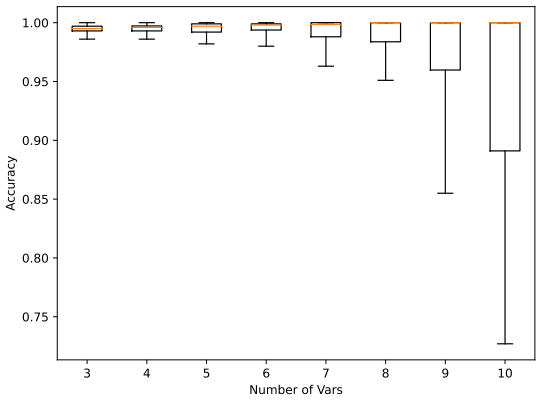

Figure 3: Accuracy of kernel classification varying the number of variables of training and test formulae, quantiles are averaged over 100 experiments.

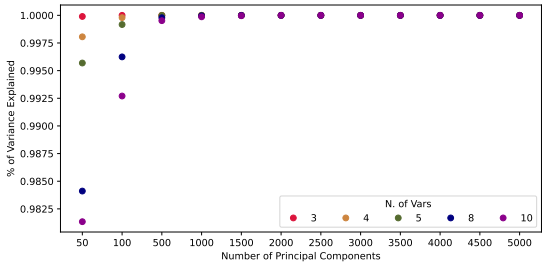

Figure 4: Proportion of explained variance against the number of principal components kept (kernel computed using 5000 formulae), for different number of variables appearing in formulae.

Moreover, we can give a deeper insight on the performance of the Boolean kernel in measuring semantic similarity between propositional formulae by showing the results of kernel classification on the satisfaction of assignments to formulae, reported in Figure 3.

As the dimensionality of the embeddings computed by the kernel is equal to the number of training formulae used to evaluate the kernel itself (i.e. it is the vector of kernel evaluations according to Equation 3 of the input formula against all the formulae in the training set), we verify if it can be reduced without significant information loss by performing dimensionality reduction in the embedding space using Kernel Principal Component Analysis (kernel PCA). We answer this question affirmatively by showing that keeping 100 components for formulae having at most 5 variables and 500 components for formulae with 6 to 10 variables retains much of the variance of the original dataset made of 5000 formulae, as shown in Figure 4.

C. More Experimental Details

C.1. Dataset Generation Algorithm

In order to construct synthetic datasets of propositional formulae we adapt the algorithm defined in [1] to our framework. It results in the recursive growing scheme for the syntax tree of each formula described in Algorithm 1, where p_{leaf} is the probability for a node to be a leaf node, n is the max number of variable indexes allowed and $Cat_{p_{\text{leaf}}}(\{\wedge, \vee, \neg, \text{VAR}\})$ denotes the categorical probability distribution assigning p_{leaf} probability to the outcome VAR (i.e. leaf node of the syntax tree) and $\frac{1-p_{\text{leaf}}}{3}$ to each operator node (i.e. internal nodes of the syntax tree).

Algorithm 1 Syntax-tree random recursive growing scheme

Require: $p_{\text{leaf}} \in (0, 1), n \in \mathbb{N}$
  $q \leftarrow \emptyset$ ▷ Queue of uncomplete nodes
  root $\leftarrow \mathcal{U}(\{\wedge, \vee, \neg\})$ ▷ Root is an operator node
  q.push(root)
  while $q \neq \emptyset$ **do**
    $N \leftarrow q$.pop() ▷ Next node to expand
    if $N \in \{\wedge, \vee\}$ **then**
      $M, R \leftarrow Cat_{p_{\text{leaf}}}(\{\wedge, \vee, \neg, \text{VAR}\}) \times Cat_{p_{\text{leaf}}}(\{\wedge, \vee, \neg, \text{VAR}\})$
      N.left_child $\leftarrow M$
      N.right_child $\leftarrow R$
      q.push(M)
      q.push(R)
    else if N is $\neg$ **then**
      $M \leftarrow Cat_{p_{\text{leaf}}}(\{\wedge, \vee, \neg, \text{VAR}\})$
      N.child $\leftarrow M$
    else ▷ N is a variable node
      N.index $\leftarrow \mathcal{U}(n)$ ▷ Variable index is sampled randomly

C.2. More Training Details

All the models described in Section 4 have been implemented in Python exploiting the PyTorch [39] library for leveraging GPU acceleration.

Then, the loss function we optimize via mini-batch gradient descent is that of Equation 1 or its variant of Equation 2 in the conditional version of our architecture. We allow the KL term of such losses to be weighted by a tunable hyperparameter $\beta > 0$, i.e. using the loss of the so-called β-VAE [40] model.

Each architecture has been trained with Stochastic Gradient Descent (SGD) on a set of 4000 different propositional formulae (generated as described in Section C.1) divided in minibatches of size 32 with an initial learning rate of 10^{-3}, using the Adam optimizer [41] and early stopping to regulate the number of training epochs (we put a validation step every 30 epochs and stop the training if validation loss does not significantly improve with a patience of 3 checkpoints; this

resulted on having the non-conditional models trained for ~ 300 epochs and the conditional ones for ~ 600 epochs).

Datasets for training and testing the models have been generated following algorithm 1, fixing $p_{\text{leaf}} = 0.4$ and varying $n \in \{3, 4, 5\}$. This resulted in sets having an average of $[10.2955, 13.6632, 17.7475]$ nodes and average depth of $[5.4687, 8.5132, 9.4419]$ for increasing n. We remark that increasing the number of variable indexes allowed increases the number of node types the network has to learn (hence further increasing the complexity of the datasets).

We tested three different encoder architectures, each consisting on a GNN with the following distinctive features:

- GRU, same as [10]: the encoder is a MPNN where, using the notation of Equation 5, M_t is a GRU and U_t a gated sum (i.e. it follows the same update scheme of the decoder in Equation 4);
- GCN: the encoder is a GCN (Equation 7) with 2 layers, unless differently specified;
- GAT: the encoder is a GAT with 3 layers having $(3, 3, 4)$ heads resp., using sum as node-wise operation of Equation 8 and concatenation as aggregator of single heads results in the internal layers and average in the final layer, unless differently specified.

All of them implement the asynchronous message passing scheme. For all the encoders, we report results for the bidirectional setting, as justified by ablation studies in Appendix D.1. To compute the mean and covariance of the approximate latent distribution $q_\phi(\cdot)$ we use two 1-layer MLPs.

The hidden size of the models is set to 250 and the latent size to 56, these hyperparameters as well the ones listed above have been tuned with the hyperparameter optimization framework Optuna [42]. As noted in [10], we also find that setting $\beta > 0.001$ as weight for the KL divergence in the loss function causes a significant drop in reconstruction accuracy.

For what concerns the semantic context vector used in the CVAE version of our model, we computed it by kernel PCA as described in Section B. Since our train and test datasets contain formulae having at most 5 variables, we keep 100 components, as justified by Figure 4.

D. More Experimental Results

In Tables 2 and 3 we report more VAE and CVAE abilities results on different datasets of increasing complexity. We remark that GAT is the best performing encoder, both for the syntactic and the semantic-conditioned task. In particular in the conditional case it is able to better capture the multimodality of each $\mathbf{y}$ w.r.t. to formulae φ and give a better characterization of the semantics in the latent space.

Concerning the latter, Figure 5 gives an insight on the relation between kernel similarity value between a formulae (φ, ψ) and distance between their kernel PCA embeddings. Being the kernel evaluated between two formulae an interpretable measure of their semantic similarity (more on this in Section B), this reported inverse correlation justifies the metrics used for evaluating CVAE abilities.

For what concerns the validity of formulae, we recall that the only case in which our decoding algorithm can produce invalid formulae is when it reaches the maximum number of nodes $\max_v$ allowed in the produced graphs, in our experiments set to 30, as described in Section 3.2.

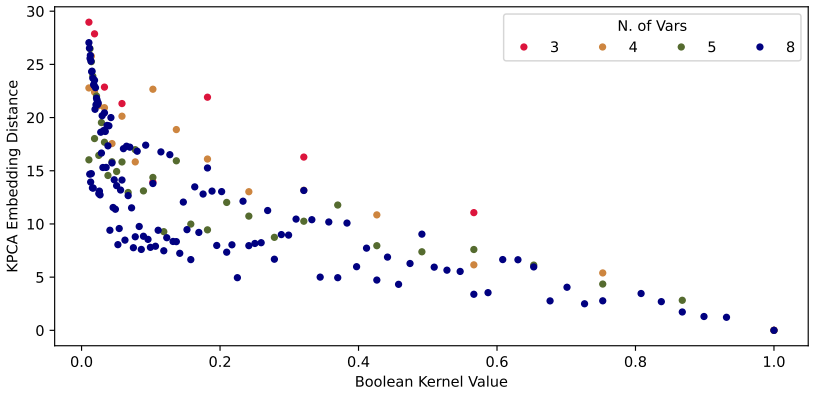

Figure 5: Correlation between kernel PCA embedding and kernel similarity between formulae, varying the number of variables for a pool of 500 formuale.

Table 2
Results of VAE abilities tests, averaged over 300 test formulae. The second line of the accuracy column for the GAT and GCN models represents the accuracy computed by considering only the most frequently decoded formula for each datapoint.

	3 var				4 var				5 var			
Encoder	Acc.	Val.	Uniq.	Nov.	Acc.	Val.	Uniq.	Nov.	Acc.	Val.	Uniq.	Nov.
GAT	**93.45**	93.42	**58.09**	**54.24**	**93.96**	93.70	**54.13**	**53.59**	**93.92**	89.38	**56.53**	**55.34**
	96.43				**97.32**				**95.42**			
GCN	91.76	**99.64**	28.06	24.83	92.03	**98.82**	31.72	29.35	91.27	**98.78**	29.91	28.32
	94.23				96.34				93.56			
GRU	88.77	98.88	15.56	15.44	83.23	86.96	14.96	13.58	82.24	75.52	10.63	19.16

Table 3
Results of CVAE abilities tests, percentages are averaged over 300 test formulae. The second line of the accuracy column for the GAT and GCN models represents the accuracy computed by considering only the most frequently decoded formula for each datapoint.

	3 var				4 var				5 var			
Encoder	Acc.	Val.	Sem. Dist.	Ker. Value	Acc.	Val.	Sem. Dist.	Ker. Value	Acc.	Val.	Sem. Dist.	Ker. Value
GAT	87.75	**98.59**	**8.819**	**0.5025**	88.11	97.55	**7.512**	**0.7692**	**87.43**	**93.72**	**6.317**	**0.7985**
	93.23				**92.76**				**92.45**			
GCN	**89.43**	98.34	9.052	0.4671	**88.87**	**99.55**	8.447	0.6115	85.35	89.14	6.808	0.6924
	94.97				92.45				91.79			

For what concerns the accuracy of reconstructed formulae, we also notice that it increases if we consider, for each of the test formulae, only the most common decoded one (which is a more realistic scenario). Accuracy in this scenario for GAT and GCN encoders is reported as second line of the corresponding model in Tables 2 and 3 for the conditioned and unconditioned case,

Figure 6
Distribution of kernel value (left) and semantic distance (right) for GAT and GCN models, on a dataset of 5 variables.

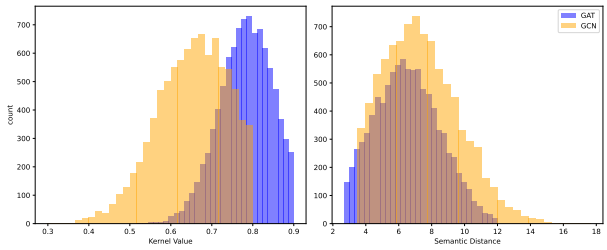

Table 4
Quantiles of the distribution of kernel value and semantic distance for GAT and GCN models, on a dataset of 5 variables.

	Kernel Values		
	Q1	Median	Q3
GAT	0.7395	0.7843	0.8276
GCN	0.5932	0.6578	0.7169
	Semantic Distances		
	Q1	Median	Q3
GAT	5.097	6.455	7.887
GCN	5.658	7.166	8.846

Table 5
VAE accuracy on several ablations of the model, averaged over 300 test formulae with 5 variables.

Encoder	Non-constrained decoding	Unidirectional	# layers 1/2/3/4/5
GAT	32.53	89.66	81.38/89.21/**93.92** /91.37/87.57
GCN	25.64	90.58	89.18/**91.27**/90.53/87.31/85.29

respectively.

In Figure 6 we show the distributions of kernel values and semantic distances computed as described earlier for LogicVAE with both GAT and GCN encoder, when trained on a dataset of 5 variables. Moreover, in Table 4 we report their first, second and third quantiles.

D.1. Ablations

In Table 5 we report the results of several ablation studies we perform on LogicVAE. They consists in the following:

1. Non-constrained decoding: this represents the most important design choice. We kept the encoding and objective function as described earlier, but we remove from the decoder the constraints imposing to follow logic syntactic rules (i.e. the ones described in Section 3.2). This leads to a dramatic decrease in reconstruction accuracy, highlighting the fundamental importance of this inductive bias.
2. Unidirectional message passing: messages are exchanged only following the original direction of the edges of the input DAG (i.e. without reverting them). For both GCN and DAG encoders, this decreases the reconstruction accuracy, but not dramatically.
3. Number of convolutional/attentional layers: we get the highest accuracy with 2-layer GCN encoder and 3-layer GAT encoder.

D.2. Hierarchical Learning of Logic Formulae

While performing experiments, we noticed that LogicVAE struggled in learning variable indexes, i.e. its reconstruction accuracy increases (and the architectures converge faster) if we provide as input a simplified AST for each input formula φ. This graph is constructed by removing variable indexes from the input described in Figure 2 and Section 3. The idea is then to build a hierarchical learning model, which first learns to reconstruct the simplified tree using the syntactic version of LogicVAE, then recovers variable indexes by a GNN architecture. We instantiated this latter model using a MPNN architecture, akin to that described in Section C.2, with node-level readout function on the leaves. We trained the model to minimize a loss involving both a classification term (namely cross-entropy loss over n classes, being n the maximum number of variable indexes allowed) and a semantic term measuring the square loss between the semantic vector $\mathbf{y}_{\varphi}$ of the ground-truth formula φ (as described in Section C.2) and the one of the reconstructed formula $\hat{\varphi}$, denoted by $\hat{\mathbf{y}}_{\hat{\varphi}}$ at each step of the training algorithm. Formally we minimize the following (for a formula φ having M leaves over n possible indexes):

$$\mathcal{L}(\varphi, \hat{\varphi}) = -\frac{1}{M} \sum_{v=1}^{M} (\sum_{i=1}^{n} y_{vi} \log \hat{y}_{vi}) + \lambda ||\mathbf{y}_{\varphi} - \hat{\mathbf{y}}_{\hat{\varphi}}||_2 \tag{10}$$

where $y_{vi}, \hat{y}_{vi}$ represent the true (resp. reconstructed) probability of leaf v to have index i, and $\lambda \in \mathbb{R}_{\geq 0}$ is a tunable parameter weighting the semantic term of the loss w.r.t. the classification one. Interestingly, we found that the loss of Equation 10 reaches a plateau in correspondence of low values of the semantic term. Indeed, we verified that the true formula φ and the reconstructed one $\hat{\varphi}$ evaluated in correspondence of the learning plateau differ on the $[10.34\%, 12.29\%, 12.48\%, 16.85\%]$ of possible configurations with $n \in [3, 4, 5, 10]$ (results with $\lambda = 0.7$ found with hyperparameter optimization, removing either the classification or the semantic term in the loss does not ameliorate the results). Hence, to some extent, a node-level MPNN learns to focus on semantically relevant parts of the input formula. We consider this results interesting towards a better understanding of the learning dynamics of GNN architectures in the context of learning logic formulae and we plan to investigate it deeper in the future.