On the Verification of Embeddings with Hybrid Markov Logic

Anup Shakya
University of Memphis
ashakya@memphis.edu

Abisha Thapa Magar
University of Memphis
thpmagar@memphis.edu

Somdeb Sarkhel
Adobe Research
somdeb@adobe.com

Deepak Venugopal
University of Memphis
dvngopal@memphis.edu

***Abstract*—The standard approach to verify representations learned by Deep Neural Networks is to use them in specific tasks such as classification or regression, and measure their performance based on accuracy in such tasks. However, in many cases, we would want to verify more complex properties of a learned representation. To do this, we propose a framework based on a probabilistic first-order language, namely, Hybrid Markov Logic Networks (HMLNs) where we specify properties over embeddings mixed with symbolic domain knowledge. We present an approach to learn parameters for the properties within this framework. Further, we develop a verification method to test embeddings in this framework by encoding this task as a Mixed Integer Linear Program for which we can leverage existing state-of-the-art solvers. We illustrate verification in Graph Neural Networks, Deep Knowledge Tracing and Intelligent Tutoring Systems to demonstrate the generality of our approach.**

I. Introduction

A typical approach to verify Deep Neural Network (DNNs) representations (or embeddings) is to evaluate them in a specific downstream task (e.g. classification). To evaluate structure in an embedding, approaches such as [1] visualize and compare the global structure of an embedding with a reference. While beneficial, such approaches are still fairly limited since they do not verify properties that combine the structure of embeddings with natural relationships that exist within the domain. For example, using the language of first-order logic, we can encode a property regarding the spread of flu as $\texttt{Friends}(x, y) \wedge \texttt{Dist}(x, y) < \tau \Rightarrow (\texttt{Flu}(x) \Leftrightarrow \texttt{Flu}(y))$, where $\texttt{Dist}(x, y)$ denotes the distance between embeddings learned by a DNN and τ is a threshold value. This property combines the geometry of the embedding (neighborhoods) with symbolic relations. In this paper, we develop a probabilistic framework to verify such properties.

Specifically, in our verification framework, we learn a *specification HMLN* (Hybrid Markov Logic Network [2]) that combine sub-symbolic representations with symbolic knowledge to represent verifiable properties. We parameterize properties with weights that represent uncertainty and learn these weights from embeddings observed from a specification DNN. To account for noise that may be inherent in specification embeddings, and to reduce variance in the parameterization, we develop a Rao-Blackwellized likelihood function that we optimize to learn the specification HMLN. We then formulate verification of a property as probabilistic inference in the specification HMLN. Specifically, similar to software/hardware verification where we compare states of variables through assert statements, here, we compare the specification embedding with a test embedding by expressing bounds on the probability of a property conditioned on the embeddings. We show that we can compute these bounds by formulating a Mixed Integer Linear Program (MILP) which can be solved by existing state-of-the-art solvers.

In our evaluation, we verify embeddings learned for varied applications illustrating the generalizability of our framework. Specifically, we i) verify semantic meaning of Graph Neural Networks embeddings and ii) verify invariance of embeddings learned through Deep Knowledge Tracing [3] and iii) verify if embeddings can transfer across tasks in an Intelligent Tutoring System using data from MATHia, a commercial provider of K-12 Math learning software.

II. Related Work

An overview of formal verification methods for DNNs is provided in [4]. Verification methods may be *complete* with stronger guarantees at the cost of poor scalability or *incomplete* that are more scalable with reduced guarantees [5]. Deductive verification methods in [6] use constraint solvers to verify DNN properties. In [7], a Neuro-Symbolic approach is proposed for verification which allows for richer specifications. Other types of approaches used for verification include abstract interpretation where the neural network layers are converted to process on abstract inputs and produce abstract outputs that over-approximates the behavior of the neural network over all real inputs [8], [9]. For verifying local robustness, [5] proposed bounds of verification through chordal graph decomposition. MILP formulations have also been used to verify robustness in [10]–[12]. However, in our case, the semantics of HMLNs allow us to specify more expressive relational properties combining symbolic and sub-symbolic representations.

III. Specification Language

To verify a model, we require a language to express properties for a model. First-order logic (FOL) is arguably the de-facto representation language that has been widely used for formal verification. In particular, constructs in FOL can be used to express symbolic and numeric terms and thus, we can

This research was supported by NSF awards #2008812 and #1934745. The opinions, findings, and results are solely the authors' and do not reflect those of the funding agencies.

represent complex properties of the model under verification by relational formulas containing discrete and continuous variables. We formalize this with the following definitions.

Definition 1. *A sub-symbolic atom is a numeric FOL term where the domain of all variables in the atom are sub-symbolic representations. A symbolic atom is a FOL term where the domain of all variables in the atom are constants.*

Definition 2. *A verifiable property is a FOL formula with terms that include both symbolic and/or sub-symbolic atoms. A ground property is one where all variables in the property have been replaced by an element from their respective domains.*

Example 1. $\texttt{Word}(w_1, x) \wedge \texttt{Word}(w_2, y) \Rightarrow \texttt{SameTopic}(x, y)$ *is a verifiable property that contains only symbolic terms.* $(D(e_w, e_{w'}) < \tau) * (\texttt{Word}(w, x) \wedge \texttt{Word}(w', x) \Rightarrow \texttt{Sentiment}(x, s))$ *is a property where* $D(e_w, e_{w'}) < \tau$ *is sub-symbolic term that represents a continuous function over sub-symbolic representations* $e_w, e_{w'}$ *(e.g. distance between embeddings).* $\texttt{Word}(W_1, X) \wedge \texttt{Word}(W_2, Y) \Rightarrow \texttt{SameTopic}(X, Y)$ *is a ground property where* W_1, W_2, X, Y *are elements from the domains of the variables* w_1, w_2, x, y *respectively.*

Definition 3. *A set of verifiable properties are parameterized as a Hybrid Markov Logic Network (HMLN) distribution where i) each ground symbolic or sub-symbolic atom is a node in the Markov network and ii) each ground property is a feature in the Markov network.*

The probability distribution of the HMLN follows a log-linear model. Specifically, given $\{f_i; w_i\}_{i=1}^k$ where f_i is a property and w_i is a real-valued weight, the distribution is given by,

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i w_i s_i(\mathbf{x})\right) \tag{1}$$

where $\mathbf{x}$ is a *world*, i.e., an assignment to all ground atoms in the HMLN and $s_i(\mathbf{x})$ is the value of the i-th formula. If f_i is a property with only symbolic atoms, then $s_i(\mathbf{x})$ is equal to the number of true groundings of f_i given the assignment $\mathbf{x}$ and if f_i is a property with continuous terms, $s_i(\mathbf{x})$ is the value of the property f_i given assignment $\mathbf{x}$.

To express properties over sub-symbolic atoms, we utilize continuous functions. While the representation is capable of handling generic continuous functions, in typical verification tasks, the most common functions are equality and inequality. These functions are represented in the HMLN distribution as *soft* equality and inequality. Specifically, the soft equality value is equal to the square of the difference between numeric terms involved in the equality. The soft inequality value is equal to the negative log-sigmoid function [2]. For instance, $D(X, Y) == D(Y, X)$ is a soft equality, where the value of this term is given by $-(D(X, Y) - D(Y, X))^2$. Similarly, $D(X, Y) < \tau$ is a soft-inequality, where the value of the term is equal $-log(1 + exp(a(\tau - D(X, Y))))$ and for $D(X, Y) > \tau$, the value is equal to $-log(1 + exp(a(D(X, Y) - \tau)))$, where τ is the threshold and a denotes the softness of the sigmoid.

IV. Specification Learning

We learn a specification HMLN by parameterizing the properties $\{f_i\}_{i=1}^k$ with weights $\mathbf{w} = \{w_i\}_{i=1}^k$ based on an *observed world*. An observed world is an assignment to all ground atoms in the HMLN. Specifically, let $\mathbf{X}$ represent all ground atoms and $\mathbf{X} = \mathbf{x}$ denote an observed world, i.e., an assignment to $\mathbf{X}$. While in a typical case, we can learn the parameters by maximizing the log-likelihood of the observed world, here, note that the observed world contains both symbolic and sub-symbolic atoms. The assignments to symbolic atoms are typically derived from domain knowledge while for sub-symbolic atoms, the values come from a specification DNN and these values may be a source of uncertainty in learning the model.

Specifically, let $\mathbf{w}^*$ be the optimal weights for verifying properties $\{f_i\}_{i=1}^k$. Let $\hat{\mathbf{w}}$ be the weights learned by optimizing the log-likelihood. Let $\sigma^2(\hat{\mathbf{w}})$ denote the variance in the estimated weights. To reduce $\sigma^2(\hat{\mathbf{w}})$, we *Rao-Blackwellize* the log-likelihood by summing out sub-symbolic atoms from the function. Specifically, the Rao-Blackwellized objective is,

$$\ell^{RB}(\mathbf{w}, \mathbf{y}, \mathbf{x}_s, \mathbf{x}_e) = \log P_{\mathbf{w}}^{RB}(\mathbf{Y} = \mathbf{y} | \mathbf{X}_e) = \log \sum_{\mathbf{x}_s} P_{\mathbf{w}}(\mathbf{y}, \mathbf{x}_s | \mathbf{X}_e) \tag{2}$$

where $\mathbf{X}_e \subseteq \mathbf{X}$ denote *evidence atoms*, i.e., symbolic atoms whose assignments are assumed to be known during inference, $\mathbf{X}_s \subseteq \mathbf{X}$ denote sub-symbolic atoms and $\mathbf{Y} = \mathbf{X} \setminus (\mathbf{X}_e \cup \mathbf{X}_s)$ denote *query atoms*, namely, atoms whose assignments are unknown and need to be predicted during inference. The partial derivative for the Rao-Blackwellized objective w.r.t the weights is as follows.

$$\frac{\partial(\ell^{RB}(\mathbf{w}, \mathbf{y}, \mathbf{x}_s, \mathbf{x}_e))}{\partial w_i} = \mathbb{E}_{\mathbf{w}, \mathbf{x}_s}[s_i(\mathbf{y}, \mathbf{x}_s, \mathbf{X}_e)] - \mathbb{E}_{\mathbf{w}, \mathbf{x}_s, \mathbf{x}_e}[s_i(\mathbf{y}, \mathbf{x}_s, \mathbf{x}_e)] \tag{3}$$

where $\mathbb{E}_{\mathbf{w}, \mathbf{x}_s}$ denotes that we compute the expectation w.r.t $\sum_{\mathbf{x}_s} P(\mathbf{y} | \mathbf{X}_e)$ and $\mathbb{E}_{\mathbf{w}, \mathbf{x}_s, \mathbf{x}_e}$ denotes that the expectation is w.r.t $\sum_{\mathbf{x}_s, \mathbf{y}} P(\mathbf{y} | \mathbf{X}_e)$. Using this, we can directly apply gradient descent to update the weights. Note that in each gradient computation, we need to estimate two expectations. Since computing them exactly is intractable, we use approximate inference to estimate each of them. Let $\hat{\mathbf{w}}^*$ denote the converged weights when we use an unbiased (or asymptotically unbiased) estimator for both $\mathbb{E}_{\mathbf{w}, \mathbf{x}_s}$ and $\mathbb{E}_{\mathbf{w}, \mathbf{x}_s, \mathbf{x}_e}$. From the Rao-Blackwell theorem, it follows that $\sigma^2(\hat{\mathbf{w}}^*) \leq \sigma^2(\hat{\mathbf{w}})$.

A. MILP Encoding

Similar to the Voted Perceptron for MLNs [13], we use MAP inference to estimate the intractable expectations. However, in our case, since we have both discrete and continuous variables in the HMLN, we formulate MAP inference as a Mixed Integer Linear Program (MILP). With this formulation, we can leverage state-of-the-art solvers to scale up weight learning even when the number of ground properties are very large. Specifically, the MILP objective is as follows.

$$\arg\max_{\mathbf{x}} \exp\left(\sum_i w_i s_i(\mathbf{x})\right) \approx \arg\max_{\mathbf{x}} \sum_i w_i s_i(\mathbf{x}) \quad (4)$$

Once we obtain the solution to the MILP, we compute the value of the i-th formula in the solution as an approximation of its expected value in the gradient computation. Since we need two expected values, we run the MILP twice. In the first case, only the evidence atoms are assumed to be known, i.e., we add constraints that assign evidence atoms to their respective values in the observed world. Thus, the MILP solution computes assignments jointly over both the sub-symbolic and query atoms. In the second case, we assume that both the evidence and sub-symbolic atoms are known, i.e., we add constraints to assign these to their respective values and the MILP computes the assignments to only the query atoms.

We encode a property (w_i, f_i) in the MILP as follows. If f_i only contains symbolic atoms, we encode an auxiliary variable a_i whose objective value is w_i and an equivalence relation $f_i \Leftrightarrow a_i$ as a linear constraint. If f_i contains both symbolic and sub-symbolic atoms, we encode each sub-symbolic atom in f_i as a continuous variable, say $a^c_{i1} \ldots a^c_{ik}$. For the symbolic sub-formula within f_i, say f_{is}, we encode a binary auxiliary variable a_i and constraint $f_{is} \Leftrightarrow a_i$. The objective value for a_i is $w_i * g(a^c_{i1} \ldots a^c_{ik})$, where $g()$ encodes the value of f_i. Note that the value of f_i may be a non-linear expression over $a^c_{i1} \ldots a^c_{ik}$. For instance, both the soft equality and inequality functions over sub-symbolic variables are non-linear functions. In such cases, we perform a relaxation by adding constraints to encode $g()$ as a piece-wise linear approximation to the true value of f_i.

B. Weight Sharing

Thus far, we assumed that each property is parameterized by a single weight. However, in practice, we may need multiple weights for the same property. For instance, suppose we are verifying an embedding learned by a deep neural network model; the geometry, sparsity, and density of points can vary across the embedding space. Thus, we may need multiple weights for the same property when defined over this space. To do this, we *share* weights over subsets of property groundings defined as hypercubes [14].

Definition 4. *A hypercube is a vector $\mathbf{H} = [\mathbf{S}_1 \ldots \mathbf{S}_m]$ where each $\mathbf{S}_i$ is a set of constants from a domain.*

Definition 5. *The projection of a hypercube $\mathbf{H} = [\mathbf{S}_1 \ldots \mathbf{S}_m]$ on a property f is the set of possible groundings of f where each variable in f is substituted by $S \in \mathbf{S}_1 \times \ldots \times \mathbf{S}_m$.*

Example 2. *Suppose we have three domains $\Delta_x = \{X_1, X_2, X_3\}$, $\Delta_y = \{Y_1, Y_2, Y_3\}$, $\Delta_z = \{Z_1, Z_2, Z_3\}$, the projection of the hypercube $[\{X_1, X_2\}, \{Y_1, Y_2\}, \{Z_1\}]$ on $\mathtt{R}(x) \wedge \mathtt{S}(x, y)$ is a set of 4 ground properties. The projection of the same hypercube on $\mathtt{T}(z) \wedge \mathtt{S}(z, y)$ is a set of 2 ground properties.*

Definition 6. *Given two hypercubes $\mathbf{H}_1$, $\mathbf{H}_2$, the merge operator $\mathbf{H}_1 \oplus \mathbf{H}_2$ generates a hypercube that contains all sets in $\mathbf{H}_1$ that have no elements in common with any set in $\mathbf{H}_2$, all sets in $\mathbf{H}_2$ that have no elements in common with any set in $\mathbf{H}_1$ and for two sets $\mathbf{S} \in \mathbf{H}_1$, $\mathbf{S}'$ in $\mathbf{H}_2$ where $\mathbf{S} \cap \mathbf{S}' \neq \emptyset$, $\mathbf{H}_1 \oplus \mathbf{H}_2$ contains $\mathbf{S} \cap \mathbf{S}'$.*

We recursively construct a set of disjoint hypercubes. First, we define a bounding hypercube that consists of all domains. In each recursive step, we select a hypercube $\mathbf{H}$ and split it into two disjoint hypercubes $\mathbf{H}^+$ and $\mathbf{H}^-$. To do this, we split a set $\mathbf{S}$ in $\mathbf{H}$ such that the average value of groundings in $\mathbf{H}^+$ is greater than in $\mathbf{H}$ and the average value of groundings in $\mathbf{H}^-$ is smaller than in $\mathbf{H}$. Since it is infeasible to evaluate all subsets in $\mathbf{S}$, we instead split $\mathbf{H}$ into $\{\mathbf{H}_i\}_{i=1}^k$, corresponding to k constants in $\mathbf{S}$. We then merge all hypercubes into $\mathbf{H}^+$ where the average value exceeds that in $\mathbf{H}$ and similarly merge all hypercubes into $\mathbf{H}^+$ where the average value is smaller than in $\mathbf{H}$.

Given a set of disjoint hypercubes, $\{\mathbf{H}_i\}_{i=1}^k$, for each property f, we project each $\mathbf{H}_i$ on f and learn a shared weight for all groundings in the projection. To formally analyze the effect of weight sharing, let $w^*_1 \ldots w^*_m$ denote the optimal parameterization, i.e., the weights that are required to verify the properties encoded in the HMLN. Suppose we instead use k hypercubes and the total number of properties is n, we learn a total of $k * n$ weights. We now define a function $Q()$ that maps an optimal weight into one of the $k * n$ learned weights so as to minimize the absolute difference across all weights. Let $max_i |w^*_i - w_i| \leq \epsilon$, where ϵ is a constant. Similar to the results derived for quantization of weights in probabilistic graphical models [15], we have the following.

Theorem 1. $\ell(\mathbf{w}^*, \mathbf{x}) - \ell(\mathbf{w}, \mathbf{x}) \leq 2kn\epsilon$

Thus, as we increase the number of hypercubes k, we can tighten the bound but at the same time, it increases the number of learnable weights. Thus, we need to achieve a trade-off between the two. In practice, given a bound on the number of hypercubes α, we continue refining the hypercubes until the number of hypercubes $k \leq \alpha$.

V. Verification

In assertion-based verification in software/hardware systems, we describe valid/invalid behavior in the form of executable assert statements where we compare variables to known valid/invalid states. Analogously, here, we compare embeddings based on properties in the specification HMLN. Specifically, we compare the embeddings generated from a specification DNN to those from a test DNN w.r.t a property. To illustrate what a prototypical verification may look like, we start with an example of a commonly occurring property of embeddings. Specifically, suppose we are given a specification DNN embedding $\mathbf{E}^*$ and want to verify if the embedding $\hat{\mathbf{E}}$ is similar, we can specify this as the following property.

$$f = (D(x, y) < \tau) * [\mathtt{Class}(x) \Leftrightarrow \mathtt{Class}(y)] \quad (5)$$

where $D(x, y)$ is the distance between embeddings of x and y, and τ is a threshold. The verification statement for this is as follows.

$$|\Omega_U(f|\mathbf{E}^*) - \Omega_U(f|\hat{\mathbf{E}})| \leq \delta_1 \wedge |\Omega_L(f|\mathbf{E}^*) - \Omega_L(f|\hat{\mathbf{E}})| \leq \delta_2 \tag{6}$$

where δ_1 and δ_2 are constants, Ω_U is an upper bound on the probability of the property in the HMLN distribution and Ω_L is a lower bound. To generalize, in order to verify a property f for a test embedding based on a specification, we compute the absolute difference between bounds on the probability distribution of f in the HMLN when conditioned on these embeddings. Next, we show that we can estimate these bounds through MAP values.

We consider a special case which we term as *constrained evidence set*. Specifically, $\{\mathbf{E}_1, \dots \mathbf{E}_k\}$ is a constrained evidence set if the evidence on symbolic atoms remain fixed in all $\mathbf{E}_i$ but evidence on the sub-symbolic atoms are allowed to vary. This models a typical situation when we are trying to verify sub-symbolic representations learned through different methods while domain knowledge which is used to specify evidence on the symbolic atoms remains fixed. Note that by conditioning an HMLN on any $\mathbf{E}_i$, we condition on all the sub-symbolic atoms and thus, we obtain a discrete conditional distribution.

Given a HMLN and constrained evidence set $\{\mathbf{E}^*, \hat{\mathbf{E}}\}$, for a ground property f in the HMLN, let $f_s = 1$ indicate that the symbolic sub-formula in f is true and $f_s = 0$ that it is false. Let (M^*_+, M^*_-) be the MAP values in the distribution conditioned on $\{\mathbf{E}^*, f_s = 1\}$ and $\{\mathbf{E}^*, f_s = 0\}$ respectively. Similarly, let $(\hat{M}_+, \hat{M}_-)$ be the MAP values in the distribution conditioned on $\{\hat{\mathbf{E}}, f_s = 1\}$ and $\{\hat{\mathbf{E}}, f_s = 0\}$ respectively. Using the log sum of exponential bounds, we can show the following result (we skip the proof for lack of space).

Theorem 2. *$|\Omega_U(f|\mathbf{E}^*) - \Omega_U(f|\hat{\mathbf{E}})| \propto |(M^*_+ - M^*_-) - (\hat{M}_+ - \hat{M}_-)|$ and $|\Omega_L(f|\mathbf{E}^*) - \Omega_L(f|\hat{\mathbf{E}})| \propto |\hat{M}_- - M^*_-|$.*

In the above theorem, we assumed that the property is a ground property. To lift verification to a first-order property f, we compute the difference in mean values of the bounds over all groundings of f. Specifically, let μ_u be the mean difference between the upper bounds of the specification and the test embedding, and μ_l be the mean difference between the lower bounds. Since the number of groundings may be very large, we estimate μ_u and μ_l from sampled groundings of f. Let $\{U^*_i\}_{i=1}^k$ and $\{L^*_i\}_{i=1}^k$ represent the upper and lower bounds for k sampled groundings of f using the specification, and $\{\hat{U}_i\}_{i=1}^k$ and $\{\hat{L}_i\}_{i=1}^k$ represent the bounds for the test embedding. The sample mean differences are computed as $\hat{\mu}_u = \frac{1}{k}\sum_{i=1}^k |U^*_i - \hat{U}_i|$ and $\hat{\mu}_l = \frac{1}{k}\sum_i |L^*_i - \hat{L}_i|$. Since the variances of the bounds computed from the test and specification embeddings may be different, we use *Welch's T-Test* [16] to estimate μ_u and μ_l from $\hat{\mu}_u$ and $\hat{\mu}_l$. Thus, with a confidence interval of $1 - \gamma$, we can verify the difference between the bounds computed from the test and specification embeddings. Note that if f contains a large number of groundings, the variance in the estimated

TABLE I: Specification HMLNs for the three tasks. The property that we verify is shaded in each task. The predicate meanings are apparent from their names. The predicate $\texttt{Dist}(e_x, e_y)$ denotes sub-symbolic atoms measuring distance between e_x, e_y which represent embeddings of x and y.

GNNs	$\texttt{Class}(x_1, c) \wedge \texttt{Neighbor}(x_1, x_2) \Rightarrow \texttt{Class}(x_2, c)$ $\texttt{Dist}(e_{x_1}, e_{x_2}) < \tau * (\texttt{Class}(x_1, c) \Leftrightarrow \texttt{Class}(x_2, c))$
KT	$(\texttt{Correct}(s, p_1) \wedge \texttt{PreRequisite}(p_1, p_2)$ $\Rightarrow \texttt{Correct}(s, p_2))$ $\texttt{Dist}(e_{s_1}, e_{s_2}) < \tau * (\texttt{Correct}(s_1, p) \Leftrightarrow \texttt{Correct}(s_2, p))$
ITS	$\texttt{Success}(s, p_1) \wedge \texttt{SameTopic}(p_1, p_2) \Rightarrow \texttt{Success}(s, p_2)$ $\texttt{Dist}(e_{s_1,p_1}, e_{s_2,p_2}) < \tau * (\texttt{Success}(s_1, p_1) \Leftrightarrow \texttt{Success}(s_2, p_2))$

difference of means could be large if we uniformly sample the groundings of f. Therefore, to reduce variance, we sample a grounding of f from each hypercube and perform verification using these samples. Recall that within each hypercube all groundings of f share the same formula weight. Thus, we cover all uniquely-weighted groundings of f in the verification and at the same time, the computational complexity of verification is bounded by the number of hypercubes.

VI. Experiments

A. Setup

We utilize Gurobi to solve the MILP problem both during the learning phase and during inference. To determine the optimal configuration for training the HMLN, we perform a grid search over hyper-parameters. Specifically, we set the learning rate for gradient descent to 0.01 and set the upper limit to number of hypercubes as 200. For the verification process, we consider a confidence interval, γ as 0.05. We refer to the DNN we are verifying as the *Network Under Verification* (NUV). Our code is available here[1].

B. GNN Verification

To verify GNNs, we set up the following experiment. We learn node embeddings from three well-known GNN architectures, Graph Convolution Networks (GCNs) [17], Graph Attention Networks (GATs) [18], and GraphSage (GS) [19] with node classification as the downstream task. We use three standard benchmark datasets, namely, Cora, Citeseer, and PubMed for these experiments.

For designing the specification DNN, we use an approach similar to cross-validation, namely, we consider a DNN architecture as the specification to verify the remaining DNNs. We use the abbreviations GCN-S, GAT-S, and GS-S to denote the DNN specification. Our specification HMLN for this task is the same for all three datasets and is shown in Table. I (with $\tau = 0.5$). Here, we encode the *homophily property* over neighbors and a property relating distances between embeddings to the classes of those nodes. Thus, a closer alignment between the semantic information encoded in the embeddings and the symbolic knowledge implies that the verifier has greater confidence in the embeddings.

[1]https://github.com/anupshakya07/verification

TABLE II: Verification results for GNNs. The t-statistic is shown in each case and the ones marked in red are those where the verification fails for the NUV, i.e., p-value ≤ 0.05 for the Welch T-Test. UB refers to Upper bound and LB refers to Lower bound.

Spec.	**NUV**	**Benchmarks**						**Noisy Benchmarks**					
		Cora		**Citeseer**		**Pubmed**		**Cora**		**Citeseer**		**Pubmed**	
		UB	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**	**LB**
GCN-S	GS	-0.692	1.21	0.427	1.33	0.447	1.27	2.227	2.71	-1.22	2.66	0.6	0.276
	GAT	-0.659	0.55	1.526	0.43	0.618	-0.56	-1.528	-1.19	1.323	1.321	1.45	2.15
GS-S	GCN	0.395	1.83	-0.55	2.722	-0.58	2.29	-0.83	-0.34	-0.722	-0.623	-0.45	-0.29
	GAT	6.25	1.37	5.77	0.13	0.3	-0.53	7.33	5.44	1.71	2.94	0.2	0.19
GAT-S	GCN	2.019	0.56	-1.53	1.21	1.8	1.28	-1.65	0.33	-2.053	2.042	1.8	1.81
	GS	-0.91	0.772	-1.56	1.56	1.763	2.13	-0.96	-1.15	-1.61	-2.19	1.76	1.81

TABLE III: Verification of Knowledge Tracing embeddings testing invariance to problem exchanges. $Student-p-n-c$ to denote that there are p problems, $n*1000$ students and c latent concepts. The t-statistic is shown in each case and the ones marked in red are those where verification fails for the NUV, i.e., the p-value ≤ 0.05 for the Welch T-Test.

Dataset	**DKT-E**		**DKT-H**		**DKT-P**	
	UB	**LB**	**UB**	**LB**	**UB**	**LB**
`Student-50-1-2`	-0.288	-0.117	0.918	0.86	1.4	1.437
`Student-100-4-5`	0.991	1.875	2.38	1.7	1.95	1.671
`Student-50-4-5`	0.561	1.12	2.13	3.51	0.981	1.163

Table II shows our results for all GNN verification tests. The values in red indicate *failed verification*. That is, when the Welch T-Test has $p \leq 0.05$ indicating that the NUV bounds (UB/LB) deviate from the specification. As seen here, for the original benchmarks, the verification passes for the benchmarks for most specifications which implies that the embeddings indeed encode semantic meaning. We then created alternate benchmarks by perturbing the original graphs in an approach similar to [20], i.e., by adding noise to the graphs such that accuracy remains approximately the same, i.e., classification accuracy cannot be used for verification. On the other hand, the verification results on the noisy benchmarks in Table II show that the semantic meaning encoded in the embeddings is significantly altered in several cases. In some cases, the meaning is preserved even in the presence of noise. For instance, when the specification is GCN and the NUV is GAT or vice-versa. One possible explanation could be that the aggregation of local neighborhoods using the attention mechanism helps in minimizing the change in embedding meaning in the presence of noise. On the other hand, GS though more scalable, since it approximates the local neighborhoods through samples, could result in a loss of semantic information in the embeddings.

C. Knowledge Tracing Verification

Deep Knowledge Tracing (DKT) [3] utilizes a DNN to learn embeddings representing student knowledge which helps us predict their future performance. To illustrate verification in this model, we use publicly available datasets provided in [3] which uses Item Response Theory (IRT) [21] to generate the student data. Specifically, in IRT, each problem is related to a latent concept, and the problems are of varying difficulty levels. The idea is to sample the responses of a student based on their skills and problem difficulty. The students' skills in a concept improve each time they encounter a problem from the same concept. We label the datasets as follows, $Student-p-n-c$ to denote that there are p problems, $n*1000$ students and c latent concepts. In this task, we verify *exchangeability* in DKT models. Specifically, the ordering of problems plays an important role in KT. Our goal in this task is to verify if DKT embeddings are *invariant to ordering of problems*.

The specification DNN is the DKT model developed in [3] where an LSTM is trained to predict the next response of a student to a problem based on a sequence of student responses. The specification HMLN is shown in Table I. The first property specifies that a problem can be solved based on their pre-requisite knowledge. Specifically, for problems p_1, p_2 we consider p_1 as a pre-requisite of p_2 if they are both from the same latent concept and p_1 is easier than p_2. In the next property, we encode that two students who have similar knowledge embeddings have similar capabilities in solving a problem.

We design the NUVs to verify if the DKT model learns similar embeddings when the ordering of problems that students work on have constraints. Specifically, in DKT-E, we learn an exchangeable LSTM using the idea in [3] where we train the model over several different orderings of the problems. In DKT-P, we constrain the ordering where, given the original ordering, we only exchange problems while preserving the prerequisite structure. Specifically, we exchange two problems p_1, p_2 from the same concept if p_2 is easier than p_1, i.e., students learn to solve progressively harder problems. In DKT-H, we do the opposite, i.e., exchange problems in the original ordering if they are from the same concept such that harder problems appear before easier problems in the ordering.

Table III presents the verification results on three datasets. As seen here, for DKT-E, the verification passes for all datasets since the exchangeable model learns from different orderings. For DKT-P, the embeddings remain similar to the specification embeddings as indicated by the verification results. For DKT-H, there is a shift in the knowledge of students since it violates the prerequisite structure as shown by our verification results.

D. ITS Verification

Intelligent Tutoring Systems (ITSs) provide feedback to students during one-on-one interactions. In this task, we verify

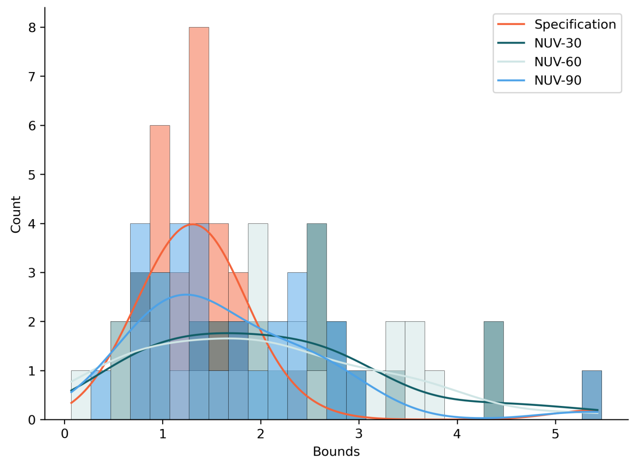

Fig. 1: Illustrating the distribution of MAP upper bounds over 50 queries for the specification and NUV models.

how well embeddings can *transfer* from learning a student's ability to providing adequate feedback which can help an ITS adapt to the student's learning. We use the Carnegie Learning MATHia 2019-20 dataset containing interactions between students and the ITS, publicly available through DataShop [22]. Our specification is a transformer model used in [23] that predicts if a student gets a step in a problem correct or not. The NUV is a DNN that predicts the level of hints for each step (level-0 being minimal hints and level-3 indicating maximum hints). Our goal is to verify if embeddings learned in the specification can *transfer* to the NUV task.

In the HMLN shown in Table I, we define a common predicate called $\texttt{Success}(s, p)$ that defines the criteria for success for a student in both the specification and the NUV DNNs. In the specification, we define $\texttt{Success}(S, P) = 1$ if the student S gets more than 75% of the steps correct in problem P. In the NUV, we define $\texttt{Success}(S, P) = 1$ if the student S uses less than 25% hints for problem P that are level-2 or above. The specification HMLN encodes a property that if a student is successful in a problem, then they are likely to be successful in other problems from the same topic. Further, we also define a property where the similarity in embeddings in the specification DNN transfers to the similarity of success in the NUV.

Here, our NUVs simulate students interacting with the ITS. Specifically, in a typical use-case, K-12 students work on different topics sequentially. Therefore, to model this, we train the NUVs over a sample of problems over different topics. A NUV trained over $t\%$ topics is denoted by NUV-t. The verification results are illustrated in the plot in Fig. 1. Here, we show the distribution of the MAP upper bounds (the lower bounds follow a similar plot, but we do not show it for lack of space) for three NUV models over all hypercubes. As seen here, the NUV bounds approach the specification bounds as we cover more topics. That is, we are able to transfer the embeddings better when we cover more topics. In the real world, this implies that the ITS can provide better hints to students if the students interact with the ITS over greater number of topics.

VII. Conclusion

We presented a general approach for verification based on a first-order probabilistic model. Specifically, we encoded verifiable properties by relating symbolic domain knowledge with sub-symbolic DNN terms and parameterized these properties as a HMLN. To perform verification, we computed bounds using MILP solvers on the probabilities of a property. We illustrated verification in our framework using GNNs, Deep Knowledge Tracing and Intelligent Tutoring Systems.

References

[1] A. Boggust, B. Carter, and A. Satyanarayan, "Embedding comparator: Visualizing differences in global structure and local neighborhoods via small multiples," in *IUI*, 2022, p. 746–766.
[2] J. Wang and P. Domingos, "Hybrid markov logic networks," in *AAAI*, 2008, p. 1106–1111.
[3] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein, "Deep knowledge tracing," in *NeurIPS*, 2015, pp. 505–513.
[4] A. Albarghouthi, *Introduction to Neural Network Verification*. verified-deeplearning.com, 2021, http://verifieddeeplearning.com.
[5] B. Batten, P. Kouvaros, A. Lomuscio, and Y. Zheng, "Efficient neural network verification via layer-based semidefinite relaxations and linear cuts," in *IJCAI*, 2021, pp. 2184–2190.
[6] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *CAV*, 2019, pp. 443–452.
[7] X. Xie, K. Kersting, and D. Neider, "Neuro-symbolic verification of deep neural networks," in *IJCAI*, 2022, pp. 3622–3628.
[8] P. Prabhakar and Z. Rahimi Afzal, "Abstraction based output range analysis for neural networks," in *NeurIPS*, 2019, pp. 15 762–15 772.
[9] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, 2019.
[10] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," *NeurIPS*, vol. 29, 2016.
[11] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *ICLR*, 2019.
[12] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of relu-based neural networks via dependency analysis," *AAAI 2020*, vol. 34, no. 04, pp. 3291–3299, 2020.
[13] P. Domingos and D. Lowd, *Markov Logic: An Interface Layer for Artificial Intelligence*. San Rafael, CA: Morgan & Claypool, 2009.
[14] P. Singla, A. Nath, and P. Domingos, "Approximate lifting techniques for belief propagation," *AAAI*, 2014.
[15] L. Chou, P. Sahoo, S. Sarkhel, N. Ruozzi, and V. Gogate, "Automatic parameter tying: A new approach for regularized parameter learning in markov networks," *AAAI*, vol. 32, no. 1, 2018.
[16] B. L. Welch, "The generalization of 'student's' problem when several different population variances are involved," *Biometrika*, pp. 28–35, 1947.
[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *ICLR*, 2018.
[19] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.
[20] A. Alchihabi and Y. Guo, "Learning robust graph neural networks with limited supervision," in *AISTATS*, 2023, pp. 8723–8733.
[21] F. Drasgow and C. L. Hulin, *Item response theory*. Consulting Psychologists Press, 1990.
[22] J. C. Stamper, K. R. Koedinger, R. S. J. de Baker, A. Skogsholm, B. Leber, S. Demi, S. Yu, and D. Spencer, "Datashop: A data repository and analysis service for the learning science community," in *AIED*, vol. 6738, 2011, p. 628.
[23] A. Shakya, V. Rus, and D. Venugopal, "Scalable and Equitable Math Problem Solving Strategy Prediction in Big Educational Data," in *EDM*, 2023, pp. 137–148.