Reparameterizing Hybrid Markov Logic Networks to handle Covariate-Shift in Representations

Anup Shakya[1] **Abisha Thapa Magar**[1] **Somdeb Sarkhel**[2] **Deepak Venugopal**[1]

[1]Department of Computer Science , The University of Memphis, TN 38152 , USA
[2]Adobe Research

Abstract

We utilize Hybrid Markov Logic Networks (HMLNs) to combine embeddings learned from a Deep Neural Network (DNN) with symbolic relational knowledge. Since a DNN may not always learn optimal embeddings, we develop a mixture model to reduce variance in the HMLN parameterization. Further, we perform inference in our model that is robust to covariate shifts that may occur in the DNN embeddings by reparameterizing the HMLN. We evaluate our approach on Graph Neural Networks and show that our approach outperforms state-of-the-art methods that combine relational knowledge with DNN embeddings when we introduce covariate shifts in the embeddings. Further, we demonstrate the utility of our approach in inferring latent student knowledge in a cognitive model called Deep Knowledge Tracing.

1 INTRODUCTION

Hybrid Markov Logic Networks (HMLNs)[Wang and Domingos, 2008] are statistical relational models that compactly represent probabilistic graphical models using first-order logic structures. They are particularly well-suited for Neuro-symbolic (NeSy) reasoning [Kautz, 2022] since they allow us to unify discrete symbolic knowledge with functions. Specifically, real-world data in critical domains such as education, healthcare, etc. often has relational structure, and using a HMLN, we can develop models where we express symbolic relationships in the data combined with deep representations (or embeddings) learned from the same data.

While representation-wise, a standard HMLN can be directly used to learn such a model, there are two main challenges in learning and inference that we address in this paper. First, real-valued functions defined over embeddings may have greater uncertainty compared to symbolic relationships. For instance, suppose e_1, e_2 are embeddings learned by a DNN for X_1, X_2 respectively, consider the product between a function over the embeddings, $f(e_1, e_2)$ and a symbolic relationship such as $\texttt{Friends}(X_1, X_2) \wedge \texttt{LikesSports}(X_1) \Rightarrow \texttt{LikesSports}(X_2)$. In this case, the symbolic relationship is directly observable in data, however, $f(e_1, e_2)$ is indirectly inferred from data through the DNN. In a typical HMLN formulation, we would assume that a single real-valued weight can parameterize the hybrid formula that combines the real-valued function with the symbolic relationship. However, when the domain of the real-valued function corresponds to embeddings learned from the DNN, this adds an extra layer of uncertainty due to embedding variability. To address this, we develop a mixture model where we combine variations of the embedding to reduce uncertainty in the parameterization.

The second challenge is related to inference. In a typical scenario, we fix the parameters of the HMLN learned from training data and then perform inference conditioned on evidence observed in test data. However, in some cases, the embeddings learned from test data could result in covariate shift, though the conditional label distribution remains invariant (in the case of discriminative learning). For instance, suppose we update the model architecture or the data during test varies even slightly, then the change in embeddings can be quite significant [Shu and Zhu, 2019]. This implies that we may not be able to utilize the exact same parameters that we learned during training to perform inference on test data. To address this, we develop a reparameterization approach that modifies the parameters learned during training utilizing the covariate shift in embeddings observed during test. Specifically, we normalize the learned parameters of the HMLN with density ratios of embeddings that occur within those formulas. However, since the exact densities are intractable to compute, we estimate them with a probabilistic classifier [Grover et al., 2019].

In our experiments, we show that using embeddings learned from Graph Neural Networks on commonly used benchmarks, we can learn a model that has a better fit (measured through conditional log-likelihood) than current state-of-the-art methods that augment statistical relational models with DNNs such as Neural PSL [Pryor et al., 2023] and DeepStochLog [Winters et al., 2022]. We show that the difference between our approach and existing methods is particularly significant when the test embedding distribution varies from the training distribution. Next, we demonstrate our approach using a cognitive model called Deep Knowledge Tracing (DKT) Piech et al. [2015] commonly used to represent student knowledge, and show that our approach outperforms the standard DKT model in the presence of covariate shift.

2 BACKGROUND

2.1 RELATED WORK

The growing area of Neuro-Symbolic (NeSy) reasoning [Raedt et al., 2020, Sarker et al., 2021] seeks to integrate DNN and symbolic models. Kautz [2022] proposed a taxonomy of NeSy models based on how the DNN and symbolic components interact with each other. Previously, Statistical Relational Models [Getoor and Taskar, 2007] was a predominant approach in unifying the representational power of first-order logic with probabilistic models to provide a general framework for uncertainty quantification in the presence of relational structure. Markov Logic Networks (MLNs) [Domingos and Lowd, 2009], Probabilistic Soft Logic (PSL) [Bach et al., 2017] and Problog [De Raedt et al., 2007] are arguably three of the most well-known statistical relational models. pLogicNet [Qu and Tang, 2019] utilizes MLNs to represent domain knowledge with first-order logic and handle uncertainty, while also incorporating knowledge graph embedding methods for efficient inference. Neural Markov Logic Networks [Marra and Kuželka, 2021] introduced an approach where instead of symbolic rules in an MLN, neural networks are used within a log-linear model. Problog was extended to DeepProbLog [Manhaeve et al., 2018], which supports both symbolic and sub-symbolic representations and inference by integrating neural networks through the use of neural predicates. [Winters et al., 2022] further extended on this and introduced DeepStochLog, which introduces neural networks into stochastic logic programs based on stochastic definite clause grammars, defining a probability distribution over derivations to enable better scalability and handling of longer sequences compared to DeepProbLog. [Maene and De Raedt, 2023] introduced DeepSoftLog, which is a framework that integrates soft-unification and probabilistic logic programming, where they use distance between embeddings instead of exact-matching of the symbolic terms for unification. More recently, PSL was extended to NeuPSL [Pryor et al., 2023] to augment DNN learning with logical rules. However, the aforementioned approaches do not account for variations in DNN representations during learning/inference, which is the focus of our work. Related to our mixture model approach, a stacking method was developed to scale-up learning in MLNs Islam et al. [2018] that combines multiple MLNs that are compressed using symmetries.

2.2 HYBRID MARKOV LOGIC NETWORKS

Markov Logic Networks (MLNs) [Domingos and Lowd, 2009] compactly represent probabilistic graphical models (PGMs) in the form of first-order logic formulas to define a distribution over possible worlds, where a world is an assignment to all the ground atoms (an atom substituted with constants) in the MLN. Hybrid Markov Logic Networks (HMLNs) generalize MLNs to include both continuous and discrete variables. Specifically, a HMLN consists of pairs $\{(F_i, \Theta_i)\}_{i=1}^n$, where F_i is a first-order formula that can contain one or more real-valued terms and Θ_i is its weight.

Each ground formula of the HMLN (substituting variables in a formula with constants from their respective domains) represents a potential function in a PGM, where the ground atoms are nodes and a clique between them represents the potential function. Symbolic ground predicates are binary random variables and real-valued ground predicates have a continuous value. The probability distribution is a log-linear model defined as follows.

$$P_\Theta(\omega) = \frac{1}{Z} \exp\left(\sum_i \Theta_i s_i(\omega)\right) \tag{1}$$

where ω is a world, i.e., an assignment to all ground predicates and Z is the *partition function*, i.e., $Z = \sum_{\omega'} \exp\left(\sum_i \theta_i s_i(\omega')\right)$. $s_i(\omega)$ is the sum of values over all groundings of F_i.

We can perform marginal inference in HMLNs using Gibbs sampling [Geman and Geman, 1984]. Specifically, given a query variable Y (we assume this to be a single variable) and observed evidence $\mathbf{X} = \mathbf{x}$ (that includes real-valued terms), we compute $P(Y = y|\mathbf{X})$ as follows. We initialize the assignments to all non-evidence variables ($\mathbf{Y}$) as $\mathbf{y}^{(0)}$. In each iteration, we pick a single variable $Y' \in \mathbf{Y}$ and sample an assignment to $Y' = y'$ from the conditional HMLN distribution $P_\Theta(Y'|\mathbf{y}_{-y'}, \mathbf{x})$ to obtain the next state of the Markov Chain $\mathbf{y}^{(1)}$. Here, $\mathbf{y}_{-y'}$ is the assignment to all variables other than Y' and therefore the conditional distribution is typically easy to compute. We estimate the marginal probabilities from samples collected after a *burn-in* period (initial samples are ignored to allow the MCMC chain to mix) using the following estimator.

$$\hat{P}(Y = y|\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} P_\theta(Y = y|\mathbf{y}_{-y}^{(t)}, \mathbf{x}) \tag{2}$$

It can be shown that as $T \to \infty$, $\hat{P}(Y = y|\mathbf{x})$ converges to the true marginal probability $P(Y = y|\mathbf{x})$.

3 APPROACH

We motivate our learning approach from a Bayesian perspective. Specifically, let f denote the optimal density function that can be learned from a dataset $\mathcal{D}$. Let $\mathcal{M}$ denote an HMLN with parameters $\Theta_{\mathcal{M}}$ when trained using $\mathcal{D}$ and Φ, where Φ corresponds to representations learned by a DNN from $\mathcal{D}$. In our case, we assume that the parameters have finite values, i.e., we do not consider hard constraints (where the weight has an infinite value) in the HMLN. We can therefore express the conditional probability over the density as follows.

$$P(f|\mathcal{D}) = \int_{\Phi}\int_{\Theta_{\mathcal{M}}} P(\Theta_{\mathcal{M}}, \Phi|\mathcal{M}, \mathcal{D}) d\Theta_{\mathcal{M}} \times f_{\Theta_{\mathcal{M}},\Phi} d\Phi \tag{3}$$

where $f_{\Theta_{\mathcal{M}},\Phi}$ is the density function computed using the HMLN. If we assume that the DNN learning is independent of the HMLN, we can simplify the above equation as follows.

$$P(f|\mathcal{D}) = \int_{\Phi}\int_{\Theta_{\mathcal{M}}} P(\Theta_{\mathcal{M}}|\Phi, \mathcal{M}, \mathcal{D}) d\Theta_{\mathcal{M}} \times P(\Phi|\mathcal{D}) \times f_{\Theta_{\mathcal{M}},\Phi} d\Phi \tag{4}$$

Clearly, computing the optimal density is hard since the weighting factors of $f_{\Theta_{\mathcal{M}},\Phi}$ require the computation of intractable probabilities. Further, since Φ is learned through a DNN, the representation may be sub-optimal which induces uncertainty when we try to learn a single parameterization for the density. To reduce this uncertainty, an approach that is used is to instead average over multiple parameterizations [Smyth and Wolpert, 1997]. Specifically, we learn a mixture over parameterizations for an HMLN based on variants of representations learned by the DNN.

3.1 MIXTURE MODEL

Let the dataset $\mathcal{D}$ be partitioned into $(\mathbf{y}, \mathbf{x})$, where $\mathbf{y}$ is an assignment on query atoms $(\mathbf{Y})$ and $\mathbf{x}$ is an assignment on evidence atoms $(\mathbf{X})$. To make equations more readable, we use the shorthand $\mathbf{y}$, $\mathbf{x}$ to represent $\mathbf{Y} = \mathbf{y}$, $\mathbf{X} = \mathbf{x}$ respectively when the context is clear. Let $\{\Phi^i\}_{i=1}^n$ denote n different DNN representations for $\mathcal{D}$. Given the HMLN structure $\mathcal{M}$, the conditional log-likelihood (CLL) of the K-component mixture model is as follows.

$$\ell(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{n} \log \sum_{j=1}^{K} \alpha_j P_{\Theta^j}(\mathbf{y}|\mathbf{x}, \Phi^i) \tag{5}$$

where $P_{\Theta^j}(\cdot)$ is the j-th parameterization of $\mathcal{M}$ and α_j is the mixture coefficient. We learn the mixture model by maximizing the negative CLL.

$$\min_{\alpha_1 \dots \alpha_K; \Theta^1 \dots \Theta^K} \sum_{i=1}^{n} -\log \sum_{j=1}^{K} \alpha_j P_{\Theta^j}(\mathbf{y}|\mathbf{x}, \Phi^i) \tag{6}$$

As is typical in mixture models, we use the EM algorithm to optimize the above objective. In the E-step, we fix the K parameterizations of $\mathcal{M}$, i.e., $\{\Theta^j\}_{j=1}^K$ and compute the probability of the query variables w.r.t each parameterization in the mixture. Specifically,

$$\gamma_{ij} = \frac{\alpha_j P_{\Theta^j}(\mathbf{y}|\mathbf{x}, \Phi^i)}{\sum_{k=1}^{K} \alpha_k P_{\Theta^k}(\mathbf{y}|\mathbf{x}, \Phi^i)} \tag{7}$$

However, note that unlike tractable models (e.g. Gaussians), computing $P_{\Theta^j}(\mathbf{y}|\mathbf{x}, \Phi^i)$ for an HMLN is computationally intractable since it requires the partition function which is $\#P$-hard. Therefore, we instead approximate the joint distribution over the query variables using Gibbs sampling. Specifically, we compute the mean-field approximation over the query variables, $\prod_{Y \in \mathbf{Y}} P_{\Theta^j}(Y|\mathbf{x}, \Phi^i)$, where $P_{\Theta^j}(Y|\mathbf{x}, \Phi^i)$ is the marginal probability over a single query variable computed from samples drawn from the HMLN with parameters Θ^j.

In the M-Step, we update the K parameterizations of the HMLN given the mixture component probabilities. To do this, we maximize the relaxed objective (with the approximated probability) as follows.

$$\min_{\alpha_1 \dots \alpha_K; \Theta^1 \dots \Theta^K} \sum_{i=1}^{n} \sum_{j=1}^{K} -\gamma_{ij} \log P_{\Theta^j}(\mathbf{y}|\mathbf{x}, \Phi^i) \tag{8}$$

Proposition 1. *If parameterized by positive weights Θ^j, the negative CLL $-\log P_{\Theta^j}(\mathbf{y}|\mathbf{x}, \Phi^i_{\mathcal{M}})$ is a convex function.*

From the above proposition, and from Jensen's inequality, it follows that the M-step optimizes a lower bound on Eq. (5). To optimize the relaxed objective, we use gradient descent. Specifically, the gradient w.r.t the k-th HMLN formula in parameterization Θ^j is as follows.

$$\frac{\partial \hat{\ell}}{\partial \theta_k^j} = \sum_{i=1}^{n} \gamma_{ij} * (s_k(\mathbf{y}|\mathbf{x}, \Phi^i) - \mathbb{E}[s_k(\mathbf{y}|\mathbf{x}, \Phi^i)]) \tag{9}$$

where $s_k(\mathbf{y}|\mathbf{x}, \Phi^i)$ is the value of the k-th formula observed in $\mathcal{D}$ and $\mathbb{E}[s_k(\mathbf{y}|\mathbf{x}, \Phi^i)]$ is its expected value based given the parameterization Θ^j. However, computing the exact expected value in Eq. (9) is intractable since it requires computation of the normalization constant. Therefore, we use the Voted Perceptron approach [Singla and Domingos, 2005] to estimate the expectation from the Maximum a Posteriori (MAP) assignment. Specifically, the MAP solution is the most probable state of non-evidence atoms in the HMLN given the evidence atoms. In our case, the MAP objective can be written as follows.

$$\arg\max_{\mathbf{y}'} \sum_{i} \sum_{k} \theta_k^i s_k(\mathbf{y}', \mathbf{x}, \Phi^i) \tag{10}$$

Existing approaches such as MaxWalkSAT [Selman et al., 1996] or ILP solvers (which we use in our experiments) can be used to solve Eq. (10) and approximately compute the MAP assignment. To estimate the expected value of the k-th formula in the gradient equation Eq. (9), we simply compute the value of the formula based on the state of its atoms in the MAP assignment. We then update the parameterizations $\theta^1 \dots \theta^K$ by multiplying the gradient with a small learning rate. Finally, we update the mixture coefficients $\alpha_k = \frac{\sum_i \gamma_{ij}}{n}$. We stop when the weights have converged to a local minima or after a fixed number of iterations.

3.2 REPARAMETERIZED INFERENCE

The marginal probability of a ground atom can be written as a ratio of partition functions. Specifically, for a ground atom Y, we can compute its marginal as follows.

$$P(Y=1|\mathbf{x},\Phi) = \sum_{i=1}^{K} \alpha_i \frac{Z_i(Y=1|\mathbf{x},\Phi)}{Z_i(Y=1|\mathbf{x},\Phi) + Z_i(Y=0|\mathbf{x},\Phi)} \tag{11}$$

where $Z_i(Y=y|\mathbf{x},\Phi)$ is the partition function of the i-th parameterization of the HMLN in the mixture conditioned on evidence atoms **x** and representation Φ. Suppose $\mathbf{y'}_{-Y}$ denotes an assignment to all atoms other than Y, the partition function sums over all possible states of $\mathbf{y'}_{-Y}$.

$$Z_i(Y=y|\mathbf{x},\Phi) = \sum_{\mathbf{y'}_{-Y}} \exp\left(\sum_j \theta^i_j \sum_k s_{jk}(\mathbf{y'}_{-Y},\mathbf{x},\Phi)\right) \tag{12}$$

While Eq. (11) yields the exact marginal probabilities, the computation is intractable ($\#P$) since we need to sum over all possible states of $\mathbf{y'}_{-Y}$. A typical approach is to use sampling methods such as Gibbs sampling to approximate the marginals. However, performing inference using the parameterizations learned by maximizing the CLL in Eq. (8) assumes that the DNN representations that we condition on during learning and inference follow the same distribution.

If the representations diverge significantly, then weights corresponding to the (locally) optimal CLL will not yield accurate uncertainty estimates during inference. DNNs can learn different representations that minimize the same empirical risk due to variations in architecture, data, hyperparameters etc. Therefore, when we condition on DNN representations during inference, we want to account for possible *covariate shift* that may have occurred in the representation. To address this, we *reparameterize* the HMLNs by importance weighting the formulas proportional to the amount of covariate shift in the DNN representation.

3.2.1 Reparameterization with Density-Ratios (DR)

Our approach to reparameterize the HMLN is based on the domain-aware MLN (DA-MLN) formalization proposed in Mittal et al. [2019]. Specifically, the idea in DA-MLNs is to scale-down the parameters of a first-order formula in the MLN based on a factor computed from its ground formulas. In contrast to regular MLNs, using reparameterization, DA-MLNs represent marginal distributions more effectively even as the domain-size (number of ground formulas) increases. In DA-MLNs, the scaling factor for reparameterization is defined as an aggregate function over the number of connections within the groundings of a first-order formula in the MLN. For completeness, we repeat the definition of DA-MLNs below.

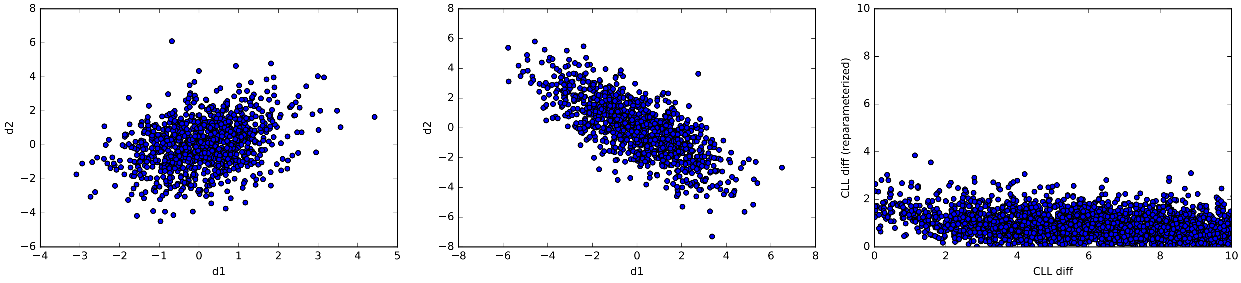
Definition 1. *Given a first-order formula F, let the variables that occur only in predicate V in F and no other predicates in F be $\bar{V}(F)$. The number of connections for V is $max(1, \prod_{x \in \bar{V}(F)} |\Delta_x|)$, where Δ_x is the domain of the variable x.*

Let formula F contain predicates $V_1 \dots V_k$. We compute the number of connections for $V_1 \dots V_k$ in F, say $C_1 \dots C_k$ and the scale-down factor is an aggregate over $C_1 \dots C_k$. Given an MLN with m formulas having parameters $\theta_1 \dots \theta_m$, if the scaling-factors for the m formulas are $w_1 \dots w_m$, the reparameterized DA-MLN marginal distribution is defined as follows.

$$\hat{P}(\mathbf{Y}=\mathbf{y}|\mathbf{X}=\mathbf{x}) = \frac{1}{Z}\exp\left(\sum_{i=1}^{m} \frac{\theta_i}{w_i} n_i(\mathbf{x},\mathbf{y})\right) \tag{13}$$

where $n_i(\mathbf{x},\mathbf{y})$ denotes the number of satisfied groundings of the i-th first-order formula given the data $(\mathbf{x},\mathbf{y})$. Note that, the typical aggregate function used in DA-MLNs to compute the scale-down factor is $\max(C_1 \dots C_k)$. In our case, we define the scale-down factor for HMLNs based on the observation that formulas contain real-valued terms with covariate shift. We motivate this with a simple example.

Example 1. *Consider a single formula $\theta : f(x,y) * (\mathtt{R}(x) \wedge \mathtt{Q}(y))$, where θ is the parameter for the formula (we assume it to be 0.1 for this example) and $f(x,y)$ is a real-valued term. Let the domain-size for all variables be equal to 3. Thus, there are 9 groundings of the formula in the HMLN. Let the real-valued term encode a* soft equality *in the HMLN, i.e. $x=y$ is defined as $-(x-y)^2$. This imposes a Gaussian penalty for deviating from the equality with the standard deviation of the Gaussian being $\sqrt{\frac{1}{2\theta}}$. Let the groundings for $f(x,y)$ during training be sampled from a 2-D Gaussian as shown in Fig 1 (a). During inference, let the groundings be drawn from the Gaussian with the same mean but a different covariance structure as shown in Fig. 1 (b). For a given world (here, we assume the world where all groundings of $\mathtt{R}(x)$, $\mathtt{Q}(x)$ are $\mathtt{True}$), we calculate the CLL of the world over all the ground atoms conditioned on the real-valued terms. The x-axis in Fig. 1 (c) shows the absolute difference between the exact CLL when groundings for $f(x,y)$ are sampled from (a) and the exact CLL when groundings for*



(a) (b) (c)

Figure 1: Illustrating reparameterization for a synthetic example. (a) 2-D Gaussian assumed to be the true distribution from which real-valued terms are sampled. (b) 2-D Gaussian which is covariate-shifted (used during inference). (c) The x-axis denotes the difference between CLLs computed using samples from (a) and (b). The y-axis denotes the difference between CLLs computed using samples from (a) and (b) after reparameterization using the density ratio.

$f(x,y)$ are sampled from (b) for 5000 different cases. The y-axis in Fig. 1 (c) shows the same CLL difference, however, this time, we reparameterize the weight θ by normalizing it with density ratios. Specifically, we compute the ratio over probability densities for each sampled grounding of $f(x,y)$ w.r.t the distributions in (a) and (b). As we see from Fig. 1 (c), the reparameterization of the HMLN reduces the difference between the CLLs by accounting for the shift in covariate structure.

Generalizing the above example, we reparameterize the HMLN with a weighting-function over the DNN representations. Specifically,

$$\hat{P}(\mathbf{Y}=\mathbf{y}|\mathbf{X}=\mathbf{x},\Phi)=\frac{1}{Z}\exp\left(\sum_{i=1}^{m}\frac{\theta_i}{w_i(\Phi)}s_i(\mathbf{x},\mathbf{y},\Phi)\right) \tag{14}$$

We want $w_i(\Phi)$ to specify the density ratio (DR) between the DNN representations observed during inference and those used to learn the parameters. However, computing the exact DR for DNNs is infeasible since the densities do not have a closed-form solution. Therefore, we instead use a *probabilistic classifier* to estimate the DR approximately. Specifically, let $\mathcal{C}$ denote a model such that $\mathcal{C}:\phi\rightarrow[0,1]$, where 1 indicates that $\phi\in\Phi$ is a DNN representation (embedding) used in inference and 0 indicates that it is an embedding used in training. We compute the weight of an embedding as follows.

$$w(\phi)=\eta\frac{\mathcal{C}(\phi)}{1-\mathcal{C}(\phi)} \tag{15}$$

where η is the ratio of the number of embeddings in training to those used in the test. Similar to the approach in Grover et al. [2019], we train $\mathcal{C}$ as a *shallow* neural network which yields calibrated probabilities. Specifically, we use cross-entropy loss to train the shallow (1-hidden layer) neural network to distinguish between embeddings used in training and those used during inference. Further, as post-processing, we rescale the logits in the neural network using the temperature scaling approach in Guo et al. [2017]. Specifically, we use a validation set to determine the scaling parameters such that the expected calibration error of the model is minimized. We use this final calibrated network to determine the reparameterization weights. The modified distribution of the HMLN is as follows.

$$\hat{P}(\mathbf{Y}=\mathbf{y}|\mathbf{X}=\mathbf{x},\Phi)=$$
$$\frac{1}{Z}\exp\left(\sum_{i=1}^{m}\sum_{\phi\in\Phi_i}\frac{\theta_i}{w(\phi)}s_i(\mathbf{x},\mathbf{y},\phi)\right) \tag{16}$$

where Φ_i is the projection of Φ on the i-th formula, i.e., $\phi\in\Phi_i$ iff ϕ occurs in at-least one grounding of the i-th formula and $s_i(\mathbf{x},\mathbf{y},\phi)$ is the value of the ground formula that contains ϕ.

Analysis. It turns out that reparameterization of the HMLN is a form of *importance weighting* that is commonly used in importance sampling to estimate expectations from intractable distributions [Neal, 2001]. Specifically, let Q denote the embedding distribution used during parameter learning and $\hat{Q}$ the distribution during inference. Let ϕ be an embedding observed during inference. To simplify notation, let us denote the value of a ground formula (in Eq. (16)) containing ϕ as $f(\phi)$. The expected value of $f(\phi)$ can be expressed as follows.

$$\mathbb{E}_Q[f(\phi)]=\mathbb{E}_{\hat{Q}}\left[\frac{Q(\phi)}{\hat{Q}(\phi)}\theta_i f(\phi)\right]=\mathbb{E}_{\hat{Q}}\left[\frac{\theta_i}{w(\phi)}f(\phi)\right]$$

Using the linearity of expectations, we see that the expected value of a first-order formula can be expressed as $\sum_{\phi\in\Phi_i}\frac{\theta_i}{w(\phi)}f(\phi)$. The estimated expectation is *asymptotically unbiased* if the ratio $\frac{Q(\phi)}{\hat{Q}(\phi)}$ is known up to a normalization constant [Liu, 2001]. However, in our case, since this ratio cannot be computed analytically, we use an approximation from the probabilistic classifier. If the approximation is close to the exact DR, the reparameterization is more accurate. Specifically, we can show the following result.

Proposition 2. *For any embedding ϕ, let $w_i^*(\phi)$ be the exact DR and $w_i(\phi)$ be the approximate DR computed by the probabilistic classifier. If the value of each ground formula is bounded between (0,1) and $|\frac{1}{w_i^*(\phi)} - \frac{1}{w_i(\phi)}| \leq \epsilon$, then $\ell^* - \ell \leq 2\epsilon m$, where m is the number of ground formulas, ℓ^* denotes the CLL reparameterized by the exact DR and ℓ denotes the CLL reparameterized by the approximate DR.*

Marginal Probabilities. Next, we compare the marginal probabilities of the reparameterized HMLN with those in a non-reparameterized HMLN. These results are obtained by extending the ones for DA-MLNs.

Proposition 3. *Given an HMLN $[\theta : f(x,y) * (\mathtt{R}(x) \vee \mathtt{S}(y))]$, where $f(x,y)$ is real-valued, if $|\Delta_x| = |\Delta_y| = n$ then for the non-reparameterized distribution, the marginal probability for a single-variable query $(P(\mathtt{R}(A))$ converges to a constant, i.e., $\lim_{n\to\infty} P(\mathtt{R}(A)) = 1$.*

Specifically, the above proposition shows that as the number of ground formulas increases, the marginal distribution tends to become independent of the embeddings, which is not useful in quantifying uncertainty in the HMLN. On the other hand, for the reparameterized distribution, we can show that the marginals are dependent on the parameters and embeddings through the following result.

Proposition 4. *Given an HMLN $[\theta : f(x,y) * (\mathtt{R}(x) \vee \mathtt{S}(y))]$, where $f(x,y)$ is real-valued, if $|\Delta_x| = |\Delta_y| = n$ and $f(x,y) = v$, if the importance weight is $1/n$ for each grounding, the marginal probability for a single-variable query $(P(\mathtt{R}(A))$ converges to a function over θ, v, i.e., $\lim_{n\to\infty} P(\mathtt{R}(A)) = \frac{1}{1+e^{\frac{-\theta * v}{2}}}$.*

Note that for both the above propositions, due to the structure of the HMLN, it turns out that we can use lifted inference rules [Gogate and Domingos, 2011] to derive the marginal probability expressions in closed form. However, it has been shown this is infeasible in the general case [Van den Broeck and Darwiche, 2013]. Thus, it follows that the exact marginals are intractable to compute and we cannot prove the above propositions for general HMLN structures. In our case, the typical HMLN hybrid formulas we use in our experiments roughly resemble the fully-connected structure $(\mathtt{R}(x) \vee \mathtt{S}(y))$, excluding the assumption of shared value among groundings. When we remove this assumption, if every grounding can have a unique value, it follows from Van den Broeck and Darwiche [2013], that the lifted inference rules do not hold and once again, the marginals cannot be computed exactly. While in this work, we do not focus on exact inference, analyzing tractable structures for reparameterized HMLNs is an interesting future direction.

Table 1: HMLN structures.

GNN	$\mathtt{Class}(+x_1, +c) \wedge \mathtt{Cites}(+x_1, x_2) \Rightarrow \mathtt{Class}(x_2, +c)$ $\mathtt{Dist}(+e_{x_1}, e_{x_2}) < \tau * (\mathtt{Class}(+x_1, +c) \Leftrightarrow \mathtt{Class}(x_2, +c))$
DKT	$(\mathtt{Correct}(+s, p_1) \wedge \mathtt{PreRequisite}(p_1, p_2) \Rightarrow \mathtt{Correct}(+s, p_2))$ $\mathtt{Dist}(+e_{s_1}, e_{s_2}) < \tau * (\mathtt{Correct}(+s_1, p) \Leftrightarrow \mathtt{Correct}(s_2, p))$

3.2.2 Mixtures of Markov Chains

We compute marginal probabilities in our model by constructing mixtures of Markov chains where components of the mixture correspond to the component HMLN distributions. Two or more Markov chains can be combined to create mixtures following the result in Tierney [1994]. Specifically, given the mixture coefficients $\alpha_1 \ldots \alpha_K$, a mixture of Markov chains is one where the *kernel* corresponding to the i-th Markov chain is applied with a probability α_i. In our case, we assume that each of the Markov chains in the mixture are constructed using Gibbs sampling. Specifically, we initialize the assignments to the non-evidence variables as $\mathbf{y}^{(0)}$. In iteration t, we pick the i-th HMLN distribution in the mixture with probability proportional to α_i and then, as in standard Gibbs sampling, we sample a variable in the HMLN from the conditional distribution given assignments to all the other variables to generate the next state $\mathbf{y}^{(t)}$. Assuming that all weights of the component HMLN are finite (i.e., there are no hard constraints that have value ∞), there are no worlds in the HMLN distribution that have 0 probabilities and therefore, each of the Markov chains induced by the Gibbs samplers are *irreducible* and *aperiodic*. Thus, from Tierney [1994], it follows that the mixture of Markov chains is irreducible and aperiodic. Therefore, we can estimate the single variable marginals based on the estimator in Eq. (2) and the estimated marginals converge to the true marginals as the number of iterations $T \to \infty$. To determine convergence, we use the Gelman-Rubin statistic [Vats and Knudson, 2021] to check if parallel chains from dispersed starting assignments to $\mathbf{y}$ converge to the same distribution.

4 EXPERIMENTS

4.1 IMPLEMENTATION

HMLNs. We used Gurobi to implement the voted perceptron parameter learning for HMLNs. We used a maximum of 100 iterations (or until convergence) to learn the mixture model. In each gradient step, we used a learning rate of 0.01. For the HMLNs, we used an approach similar to the approach used in MLNs to learn multiple parameters for a formula instead of a single weight for all groundings since this limits the type of distributions that we can represent. Specifically, in MLNs a variable that is indicated with a "+" sign implies that for each grounding of that variable, we learn a separate weight. Thus, we can control the number

Table 2: The top table shows results when the train and test embeddings are generated using GCNs based on the original graph benchmarks. The bottom table shows results for benchmarks when we introduce covariate shift in the test embeddings. Time shown indicates time taken for inference. Results (mean and standard deviation over 5 runs of the experiment) are shown for 1000 test nodes in the benchmarks. To compute accuracy for a test node, we select the class that has the maximum marginal probability corresponding to that node.

Experiments	**Cora**			**Citeseer**		
	CLL	**Accuracy (%)**	**Time (secs)**	**CLL**	**Accuracy (%)**	**Time (secs)**
DeepStochLog	-0.74±0.2	72.8±1.2	155.61±6.26	-1.2±0.1	65.5±2.8	83±11.42
PSL	-0.64±0.11	72.84±2.15	**4.56**±0.29	-0.61±0.10	51.9±1.2	4.78±1.24
NeuPSL	-0.28±0.06	**81.2**±0.76	6.5±0.4	-0.34±0.08	**68.48**±1.14	**3.71**±0.08
HMLN	-0.23±0.08	65.43±3.35	49.58±1.5	-0.11±0.05	59.95±2.10	47.27±1.3
MIX-HMLN	**-0.21**±0.08	79.37±1.15	151±2.1	**-0.09**±0.06	65.14±1.8	146.1±3.3
	Cora (covariate shifted)			**Citeseer (covariate shifted)**		
	CLL	**Accuracy (%)**	**Time (secs)**	**CLL**	**Accuracy (%)**	**Time (secs)**
DeepStochLog	-0.64±0.06	49.3±0.12	181.26±8.73	-0.71±0.04	28.8±0.5	74.4±8.78
PSL	-0.97±0.04	65.92±1.98	**4.05**±0.21	-1.10±0.04	40.14±1.31	**3.38**±0.27
NeuPSL	-0.62±0.02	66.54±1.73	7.37±0.19	-0.55±0.13	36.2±2.68	5.64±0.24
HMLN	-0.29±0.11	58.67±3.23	50.16±1.6	-0.21±0.09	53.52±3.88	48.16±1.5
HMLN(DR)	-0.166±0.07	58.92±1.79	48.98±2.1	-0.29±0.08	64.31±2.14	47.14±1.6
MIX-HMLN	**-0.149**±0.08	**77.028**±1.12	178.5±3.15	**-0.11**±0.09	**64.93**±2.21	165.65±2.73

of groundings with shared weights, and in our case, we use a single "+" variable in each of our HMLN formulas. We assumed HMLN semantics for real-valued features over embeddings. Specifically, to represent the soft inequality, $\alpha < t$, we use the log-sigmoid function, $-\log(1 + e^{(\alpha - t)})$.

Probabilistic Classifier. For the probabilistic classifier, we used a calibrated neural network to estimate the DR. Specifically, we trained a single-hidden layer network to distinguish between training and test embeddings with a cross-entropy loss function. To improve calibration, we used the Temperature Scaling approach [Guo et al., 2017] on the output probabilities of the classifier.

Gibbs Sampling. For inference, we implemented the mixture of Markov chains with 10 parallel Markov chains with a total of 100K samples. We determined convergence using the PSRF (potential scale reduction factor) of the Gelman-Rubin diagnostic [Gelman and Rubin, 1992]. PSRF is a standard diagnostic method to assess if the sampler has converged by comparing within chain variance to variance across parallel MCMC chains. We used a burn-in period of 5K samples after which we used the samples to compute the marginal probabilities.

We implemented our approach in Google Cloud with a Tesla T4 GPU (16GB). Our implementation is available here [1].

4.2 GRAPH EMBEDDINGS

We used two benchmark datasets, Cora and Citeseer, where the DNN learns embeddings to perform node classification. The HMLN structure is shown in Table 1. The first property specifies the homophily relation; nodes that are connected in the graph have the same class. The second formula specifies that embeddings whose distance is smaller than τ (with $\tau = 0.2$) have similar classes. The soft inequality function is used to specify the real-valued feature. Since there are n^2 groundings in the hybrid formula, we reduce this by eliminating groundings where the distance between embeddings is large since the grounding has a very small value.

To learn the mixture model, we need to generate variants of embeddings. To do this, we use the dropout [Gal et al., 2017] mechanism. Specifically as shown in Gal and Ghahramani [2016], dropout is a Bayesian approximation of a Gaussian process. Thus, using dropout, we can sample from the family of embeddings generated by the DNN. To learn the mixture model, we set the initial dropout value to 0.1 and increment it by 0.05. For each value of dropout, we learn a different embedding, and we used 10 components in our mixture model.

We compare our approach with DeepStochLog [Winters et al., 2022], Probabilistic Soft Logic (PSL) [Bach et al., 2017] and NeuPSL [Pryor et al., 2023]. PSL is a purely statistical relational model, while DeepStochLog and NeuPSL are both state-of-the-art extensions of well-known statistical relational models (PSL and ProbLog respectively) to incor-

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

Table 3: Ablation Study. We vary the train and test models used to learn the embeddings. For the results on GCN (Test), the GAT model is used for training the HMLN and for GAT (Test), the GCN is used in training. We show the mean and standard deviation over 5 runs.

Method	Cora				Citeseer			
	GCN (Test)		GAT (Test)		GCN (Test)		GAT (Test)	
	CLL	Accuracy (%)	CLL	Accuracy (%)	CLL	Accuracy (%)	CLL	Accuracy (%)
HMLN	-0.28±0.18	61.55±1.26	-0.35±0.13	74.53±1.13	-0.20±0.15	57.26±2.19	-0.14±0.16	75.51±0.1
HMLN + MIX	-0.22±0.09	65.34±1.26	-0.28±0.11	68.35±0.86	-0.15±0.09	62.86±1.06	-0.12±0.9	73.54±2.06
HMLN + MIX + DR	**-0.18**±0.05	**81.27**±0.29	**-0.2**±0.06	**78.9**±0.37	**-0.14**±0.03	**64.23**±0.89	**-0.09**±0.05	**76.29**±0.83

porate DNN representations. In each case, we compare the conditional log-likelihood (CLL) scores on the set of test nodes. We perform marginal inference on test nodes and similar to prior approaches, we approximate the test CLL as the log average of the marginals computed over the test nodes. We run 5 experiments and compute the mean and standard deviation in the test CLL estimates. We also compare between 3 variants of our approach as follows. HMLN: we learn a single HMLN model, HMLN(DR): we learn a single HMLN model but perform reparameterization during inference and MIX-HMLN: we learn a mixture of HMLNs and perform reparameterization during inference.

In the first set of experiments, we learn the HMLN using embeddings from Graph Convolutional Networks (GCNs) [Kipf and Welling, 2017]. The homophily property that the HMLN encodes is aligned with the representation learned by a GCNs [Ma et al., 2022]. Our results showing the CLL on test nodes are shown in Table 2. MIX-HMLN has the largest CLL compared to the other approaches. This indicates that MIX-HMLN has the smallest uncertainty in inferring the alignment between the embeddings and the relational property. NeuPSL had a slightly higher accuracy in labeling the test nodes, however, it also has a higher uncertainty (as indicated by the smaller CLL).

Next, we introduce covariate shifts in the test embeddings using an approach similar to Alchihabi and Guo [2023]. Specifically, we remove edges and add Gaussian noise to the node features to create a *perturbed* graph. We learn embeddings (with covariate shift) by training the GCN on the perturbed graph. The GCN accuracy of test node classification on the perturbed graph is similar to the accuracy on the original graph (the difference in accuracy was less than 1%). Thus, the embeddings from the perturbed graph encode the same relationships as the embeddings on the original graph.

Our results are shown in Table 2. As shown here, MIX-HMLN outperforms all the other methods in CLL scores by significantly higher margins in this case since we account for the covariate shift during inference. As shown in the table, our approach which is a mixture model takes longer to run as compared to the other single model approaches. At the same time, the mixture reduces uncertainty which in the context of learning using embeddings is important since deep models that generate these embeddings may converge to distinct local minima. Thus, we trade-off a loss of efficiency for lower uncertainty (evidenced by the higher CLL scores). In future, we plan to develop more efficient inference methods based on activating specific components in the mixture similar to approaches used in mixture of experts.

Ablation Study. Here, we introduce covariate shift by modifying the DNN architecture during inference. Specifically, we use GCNs and Graph Attention Networks (GATs) [Velickovic et al., 2018] where one of them generates training embeddings and the other one generates test embeddings. Table 3 shows our results for the three variants of our approach. As seen here, the CLL consistently increases when we add the mixture model learning and further increases when we add the reparameterization in each case.

4.3 DEEP KNOWLEDGE TRACING

Deep Knowledge Tracing (DKT) [Piech et al., 2015] uses DNNs to learn dense embeddings representing student skills over latent concepts. Specifically, DKT is a Sequence2Sequence model trained over observations that simulate exercises of varying difficulty that students work. To train the model, the exercises are generated using Item Response Theory (IRT) [Drasgow and Hulin, 1990] (details in the Appendix B). The HMLN for this task shown in Table 1 represents *pre-requisite relational structure*, i.e., the first formula specifies that if a student gets an exercise correct then they have acquired skills corresponding to pre-requisites associated with that exercise. The second formula is a hybrid formula that relates student performance to their DKT embeddings. To learn the mixture model, we learn DKT embeddings over multiple problem orderings but we maintain the same pre-requisite structure over problems in each ordering. Specifically, if X is pre-requisite to Y, then X must appear before Y in the problem ordering. We used a total of 5 different orderings to learn the mixture model.

For training embeddings, we use data with problem difficulty values (encodes in the IRT model parameters) ranging from 1 to 3. For the test embeddings, we use difficulty values from 4 to 5. Thus, we simulate the condition when the students work on new exercises of increasing difficulty. This

Table 4: (p, n, c) denotes that there are p problems, $n * 1000$ students and c latent concepts are used by the IRT model to generate the training data. Results show F1-score for predictions made on 200 test problems on all the students in the training data.

Training Dataset	DKT	HMLN	MIX-HMLN
$(50, 1, 2)$	**0.80**	0.7	**0.80**
$(75, 2, 3)$	0.76	0.67	**0.78**
$(100, 4, 5)$	0.74	0.57	**0.78**
$(200, 5, 5)$	0.62	0.54	**0.72**
$(400, 8, 10)$	0.71	0.56	**0.74**

induces a covariate shift in the DKT embeddings learned for test data. We compute the marginal probability over the atoms $\texttt{Correct}(S, P)$, where S, P represents the student and problem respectively. Table 4 compares the F1-scores of predicting student performance (we threshold the probability at 0.5) with the original DKT model over all students for all the test problems. As seen here, MIX-HMLN outperforms DKT across all settings. This illustrates that utilizing pre-requisite relational structure combined with reparameterization in the presence of covariate shift improves generalization when the number of problems, number of latent concepts and the number of students in the data grows larger.

5 LIMITATIONS

One of the main limitations with our approach is that we assume prior knowledge is known (and reasonably correct) and can be expressed in first-order logic. A second limitation is that we assume embeddings in training and testing are generated by models optimizing the same function (i.e., for the same downstream task). From a practical perspective, as is the case with mixture models in general, our approach takes longer than methods that utilize a single model. We also assume that our HMLN does not contain hard constraints (formulas that have infinite weights) in the current work.

6 CONCLUSION

We used HMLNs to combine embeddings learned from a DNN with symbolic relational knowledge. However, when we use representations from different DNNs during training and testing, the embeddings may have covariate shift even if the two DNNs are optimizing the same function. We developed an approach to learn HMLNs assuming a covariate shift in embeddings using two main ideas. First, instead of a single model, we learned a mixture model to reduce uncertainty in the learned representation. Next, we reparameterized the learned HMLN weights to account for covariate shift in embeddings during inference. To determine this shift efficiently, we used a calibrated probabilistic classifier that estimates the density ratio between embeddings in the training and test distributions. We evaluated our approach on Graph Neural Networks and also within a well-known cognitive model called Deep Knowledge Tracing that infers student knowledge over time. We illustrated that in the presence of relational structure and covariate shift, our approach outperforms state-of-the-art methods.

In the future, we plan to extend our approach to verify properties of embeddings. Specifically, we would want to understand if embeddings align with symbolic relational properties that can be specified by domain experts. We plan to apply this approach in cognitive models of student learning, where explainability of embeddings is important to have practical applicability. Further, we also plan to develop more scalable solutions for inference within our model. As mentioned earlier, there is a trade-off between inference efficiency and lower uncertainty. There are two potential directions that could be explored to improve scalability. First, using a mixture of experts has been very successful in DNNs, including Large Language Models, and we can adapt similar approaches for computational efficiency within our model. Next, there has been substantial research in *lifted inference* methods in Markov Logic Networks Gogate and Domingos [2011] where the idea is to leverage exact/approximate symmetries to improve scalability in standard inference approaches such as Gibbs sampling Venugopal and Gogate [2012]. We plan to adapt some of these approaches to scale up inference in our HMLN model.

7 ACKNOWLEDGMENTS

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

References

Abdullah Alchihabi and Yuhong Guo. Learning robust graph neural networks with limited supervision. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, pages 8723–8733, 2023.

Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research (JMLR)*, 18(1):1–67, 2017.

Albert T. Corbett and John R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4:253–278, 1994.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen.

Problog: a probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, page 2468–2473, 2007.

P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, San Rafael, CA, 2009.

F. Drasgow and C. L. Hulin. *Item response theory*. Consulting Psychologists Press, 1990.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Andrew Gelman and Donald B Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992.

S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007. ISBN 0262072882.

Vibhav Gogate and Pedro Domingos. Probabilistic theorem proving. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 256–265, 2011.

Aditya Grover, Jiaming Song, Ashish Kapoor, Kenneth Tran, Alekh Agarwal, Eric J Horvitz, and Stefano Ermon. Bias correction of learned generative models using likelihood-free importance weighting. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1321–1330, 2017.

Mohammad Maminur Islam, Somdeb Sarkhel, and Deepak Venugopal. Learning mixtures of mlns. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 6359–6366, 2018.

Henry Kautz. The third ai summer: Aaai robert s. engelmore memorial lecture. *AI Magazine*, 43(1):105–125, Mar. 2022. doi: 10.1002/aaai.12036.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Publishing Company, Incorporated, 2001.

Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *International Conference on Learning Representations*, 2022.

Jaron Maene and Luc De Raedt. Soft-unification in deep probabilistic logic. In *Advances in Neural Information Processing Systems*, volume 36, pages 60804–60820. Curran Associates, Inc., 2023.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Giuseppe Marra and Ondřej Kuželka. Neural markov logic networks. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161, pages 908–917, 2021.

Happy Mittal, Ayush Bhardwaj, Vibhav Gogate, and Parag Singla. Domain-size aware markov logic networks. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89, pages 3216–3224, 2019.

Radford M Neal. Annealed importance sampling. *Statistics and computing*, 11:125–139, 2001.

Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Yang Wang, and Lise Getoor. Neupsl: Neural probabilistic soft logic. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4145–4153, 2023.

Meng Qu and Jian Tang. Probabilistic logic neural networks for reasoning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Luc de Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4943–4950, 2020.

Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence. *AI Communications*, 34(3):197–209, 2021.

B. Selman, H. Kautz, and B. Cohen. Local Search Strategies for Satisfiability Testing. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, pages 521–532. American Mathematical Society (AMS), 1996.

Hai Shu and Hongtu Zhu. Sensitivity analysis of deep neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4943–4950, Jul. 2019.

Parag Singla and Pedro Domingos. Discriminative training of markov logic networks. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, page 868–873, 2005.

Padhraic Smyth and David H. Wolpert. Stacked density estimation. In *Neural Information Processing Systems Conference*, pages 668–674, 1997.

Luke Tierney. Markov Chains for Exploring Posterior Distributions. *The Annals of Statistics*, 22(4):1701 – 1728, 1994.

Guy Van den Broeck and Adnan Darwiche. On the complexity and approximation of binary evidence in lifted inference. In *Advances in Neural Information Processing Systems*, volume 26, 2013.

Dootika Vats and Christina Knudson. Revisiting the gelman–rubin diagnostic. *Statistical Science*, 36(4):518–529, 2021.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR*, 2018.

D. Venugopal and V. Gogate. On lifting the gibbs sampling algorithm. In *Advances in Neural Information Processing Systems*, pages 1664–1672, 2012.

Jue Wang and Pedro Domingos. Hybrid markov logic networks. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 2, page 1106–1111, 2008.

Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):10090–10100, Jun. 2022. doi: 10.1609/aaai.v36i9.21248.

APPENDIX

A PROOFS

Proposition 1. *If parameterized by positive weights Θ^j, the negative Conditional Log-Likelihood $-\log P_{\Theta^j}(\mathbf{y}|\mathbf{x}, \Phi^i_{\mathcal{M}})$ is a convex function.*

Proof.

$$-\log P_{\Theta^j}(\mathbf{y}|\mathbf{x}, \Phi^i) = \log Z - \sum_k \theta^j_k s_k(\mathbf{x}, \mathbf{y}, \Phi^i)$$

Note that in this case, we partition the ground atoms in the HMLN into query and evidence atoms and we assume that Φ^i is always observed as evidence. Thus,

$$\log Z = \log\left(\sum_{\mathbf{y}'} \exp\left(\sum_k \theta^j_k s_k(\mathbf{y}', \mathbf{x}, \Phi^i\right)\right)$$

Thus, since $\log Z$ is a log-sum over exponentials, the result follows. □

Proposition 2. *For any embedding ϕ, let $w^*_i(\phi)$ be the exact DR and $w_i(\phi)$ be the approximate DR computed by the probabilistic classifier. If the value of each ground formula is bounded between (0,1) and $|\frac{1}{w^*_i(\phi)} - \frac{1}{w_i(\phi)}| \leq \epsilon$, then $\ell^* - \ell$ $\leq 2\epsilon m$, where m is the number of ground formulas, ℓ^* denotes the CLL reparameterized by the exact DR and ℓ denotes the CLL reparameterized by the approximate DR.*

Proof.

$$
\begin{aligned}
\ell^*(\mathbf{y}, \mathbf{x}, \Phi) &= \left(\sum_{i=1}^{m} \frac{\theta_i}{w^*_i(\Phi)} s_i(\mathbf{x}, \mathbf{y}, \Phi)\right) - \log Z \\
\ell^*(\mathbf{y}, \mathbf{x}, \Phi) &= \left(\sum_{i=1}^{m} \frac{\theta_i}{w^*_i(\Phi)} s_i(\mathbf{x}, \mathbf{y}, \Phi)\right) - \log \sum_{\mathbf{x}, \mathbf{y}', \Phi} \left(\sum_{i=1}^{m} \frac{\theta_i}{w^*_i(\Phi)} s_i(\mathbf{x}, \mathbf{y}', \Phi)\right) \\
&\leq \left(\sum_{i=1}^{m} (\frac{\theta_i}{w_i(\Phi)} + \epsilon) s_i(\mathbf{x}, \mathbf{y}, \Phi)\right) - \log \sum_{\mathbf{x}, \mathbf{y}', \Phi} \left(\sum_{i=1}^{m} (\frac{\theta_i}{w_i(\Phi)} - \epsilon) s_i(\mathbf{x}, \mathbf{y}', \Phi)\right) \\
&= \sum_{i=1}^{m} \epsilon s_i(\mathbf{x}, \mathbf{y}, \Phi) + \left(\sum_{i=1}^{m} \frac{\theta_i}{w_i(\Phi)} s_i(\mathbf{x}, \mathbf{y}, \Phi)\right) - \log \sum_{\mathbf{x}, \mathbf{y}', \Phi} \left(\sum_{i=1}^{m} (\frac{\theta_i}{w_i(\Phi)}) s_i(\mathbf{x}, \mathbf{y}', \Phi) - \log \exp \sum_{i=1}^{m} \epsilon s_i(\mathbf{x}, \mathbf{y}', \Phi)\right) \\
&= 2\epsilon m + \ell(\mathbf{y}, \mathbf{x}, \Phi)
\end{aligned}
$$

□

Proposition 3. *Given an HMLN $[\theta : f(x, y) * (\mathtt{R}(x) \vee \mathtt{S}(y))]$, where $f(x, y)$ is real-valued, if $|\Delta_x| = |\Delta_y| = n$ then for the non-reparameterized distribution, the marginal probability for a single-variable query $(P(\mathtt{R}(A))$ converges to a constant, i.e., $\lim_{n\to\infty} P(\mathtt{R}(A)) = 1$.*

Proof.

$$P(\mathtt{R}(A)) = \frac{1}{1 + \frac{Z_{\mathtt{R}(A)=0}}{Z_{\mathtt{R}(A)=1}}}$$

Using the lifted inference rules from Gogate and Domingos [2011], we write the two partition functions as follows.

$$
\begin{aligned}
Z_{\mathtt{R}(A)=1} &= e^{\theta * v * n} * 2^n \\
Z_{\mathtt{R}(A)=0} &= (1 + e^{\theta * v})^n \\
\lim_{n\to\infty} \frac{Z_{\mathtt{R}(A)=0}}{Z_{\mathtt{R}(A)=1}} &= \frac{(1 + e^{\theta * v})^n}{e^{\theta * v * n} * 2^n} = 0 \\
\lim_{n\to\infty} P(\mathtt{R}(A)) &= \frac{1}{1 + 0} = 1.
\end{aligned}
$$

□

Proposition 4. *Given an HMLN $[\theta : f(x,y) * (\mathtt{R}(x) \vee \mathtt{S}(y))]$, where $f(x,y)$ is real-valued, if $|\Delta_x| = |\Delta_y| = n$ and $f(x,y) = v$, if the importance weight is $1/n$ for each grounding, the marginal probability for a single-variable query $(P(\mathtt{R}(A))$ converges to a function over θ, v, i.e., $\lim_{n\to\infty} P(\mathtt{R}(A)) = \frac{1}{1+e^{\frac{-\theta*v}{2}}}$.*

Proof.

$$P(\mathtt{R}(A)) = \frac{1}{1 + \frac{Z_{\mathtt{R}(A)=0}}{Z_{\mathtt{R}(A)=1}}}$$

Using the lifted inference rules from Gogate and Domingos [2011], we write the two partition functions as follows.

$$Z_{\mathtt{R}(A)=1} = e^{(\theta*v/n)*n} * 2^n = e^{\theta*v} * 2^n$$

$$Z_{\mathtt{R}(A)=0} = (1 + e^{\theta*v/n})^n$$

$$\lim_{n\to\infty} \frac{Z_{\mathtt{R}(A)=0}}{Z_{\mathtt{R}(A)=1}} = \frac{(1 + e^{\theta*v/n})^n}{e^{\theta*v} * 2^n} = e^{-\theta*v/2}$$

$$\lim_{n\to\infty} P(R(A)) = \frac{1}{1 + e^{-\theta*v/2}}$$

□

B DEEP KNOWLEDGE TRACING

Knowledge Tracing [Corbett and Anderson, 1994] is a classical cognitive model that models student knowledge over time to encode latent student skills. Specifically, students work out problems and the model observes if the student answered a problem correctly/incorrectly to model knowledge acquired by the student over time. Knowledge tracing aims to model knowledge acquired by the student so that we can use this to predict how they may perform in future problems. This can be used in several applications such as to develop interventions targeting specific areas of deficiency, improve student engagement, alternate strategies that may be used to teach a student, etc. Bayesian Knowledge Tracing (BKT) [Corbett and Anderson, 1994] is a classical approach for knowledge tracing that uses a Hidden Markov Model to learn from temporal data. Specifically, in BKT, a student's knowledge is represented by a set of latent variables. As the student answers exercises related to specific *skills*, BKT updates the latent variable probabilities based on the correctness (or incorrectness) of their answers. Deep Knowledge Tracing (DKT) [Piech et al., 2015] leverages DNNs to learn dense embeddings representing student skills. Specifically, DKT is a Sequence2Sequence model trained over observations that simulate exercises that students work on of varying difficulty. Specifically, knowledge over skills is represented by the hidden layer in the Sequence2Sequence model. The model is trained over sequential observations that simulate exercises that students work on of varying difficulty. To train the model, the exercises are generated using Item Response Theory (IRT) [Drasgow and Hulin, 1990]. Specifically, given parameters α, β that represent student skill in a specific concept and exercise difficulty respectively, the probability that the student completes the exercise correctly is $P(correct|\alpha, \beta) = c + \frac{1-c}{1+\exp(\beta-\alpha)}$, where c is the probability of a random guess (which is set to 0.25). The dataset initializes the difficulty level for each exercise and also sets initial skill levels for each student. We used difficulty levels ranging from 1 to 5, with 5 being the most difficult. The students' skills are updated over time as they encounter more exercises related to the same concept.

C ALGORITHMS

The algorithm to train the mixture model is summarized in Algorithm. 1. The algorithm for performing marginal inference is summarized in Algorithm 2.

D MORE RESULTS

The tables 5 - 8 show additional results on the comparison of the log-likelihoods for the different components while learning the Mix-HMLN. These results are for inference on the Cora dataset. Fig. 2 shows the reliability diagrams for the calibrated neural network which we use to compute the importance weights.

Table 5: Conditional Log-Likelihood on the original Cora graphs for the individual HMLN components.

Experiments	Cora			
	CLL	Avg. Marginals	Accuracy (%)	Time (secs)
HMLN comp 1	-0.21±0.08	0.83	55.43±3.35	49.58±1.5
HMLN comp 2	-0.34±0.04	0.71	56.37±2.13	50.24±2.6
HMLN comp 3	-0.33±0.05	0.71	55.22±1.05	49.87±1.9
HMLN comp 4	-0.27±0.08	0.78	60.28±2.1	52.4±2.05
HMLN comp 5	-0.39±0.12	0.87	69.53±2.3	51.2±1.7
HMLN comp 6	-0.26±0.08	0.71	64.75±3.88	51.85±2.19
HMLN comp 7	-0.28±0.08	0.70	59.8±1.88	52.16±3.27
HMLN comp 8	-0.35±0.08	0.75	60.54±3.73	52.62±1.59
HMLN comp 9	-0.27±0.04	0.75	57.48±3.46	51.74±1.34
HMLN comp 10	-0.30±0.04	0.74	59.97±1.6	52.72±1.79

Table 6: Conditional Log-Likelihood on the original Citeseer graphs for the individual HMLN components.

Experiments	Citeseer			
	CLL	Avg. Marginals	Accuracy (%)	Time (secs)
HMLN comp 1	-0.10±0.05	0.81	59.95±2.10	47.27±1.3
HMLN comp 2	-0.16±0.04	0.77	59.32±2.23	44.3±1.6
HMLN comp 3	-0.12±0.05	0.80	57.97±2.19	49.11±1.1
HMLN comp 4	-0.12±0.04	0.80	58.45±2.02	49.2±2.1
HMLN comp 5	-0.16±0.07	0.76	61.33±0.79	47.1±2.1
HMLN comp 6	-0.17±0.06	0.80	58.80±1.34	48.51±1.6
HMLN comp 7	-0.13±0.05	0.78	61.56±2.30	48.86±1.8
HMLN comp 8	-0.12±0.07	0.78	60.35±2.23	48.66±1.9
HMLN comp 9	-0.16±0.07	0.79	59.87±1.47	49.81±1.4
HMLN comp 10	-0.15±0.05	0.76	58.84±1.21	49.44±1.3

Table 7: Conditional Log-Likelihood on the covariate shifted Cora graph for the individual HMLN components with reparameterization.

Experiments	Noisy Cora			
	CLL	Avg. Marginals	Accuracy (%)	Time (secs)
HMLN comp 1	-0.166±0.07	0.87	58.92±1.79	48.98±2.1
HMLN comp 2	-0.19±0.6	0.80	56.67±2.23	49.65±1.8
HMLN comp 3	-0.18±0.07	0.79	59.72±2.19	48.5±2.3
HMLN comp 4	-0.17±0.06	0.81	60.15±1.88	52.2±1.6
HMLN comp 5	-0.19±0.04	0.75	63.22±1.24	50.21±1.5
HMLN comp 6	-0.16±0.03	0.73	60.20±1.37	50.92±1.4
HMLN comp 7	-0.15±0.04	0.79	63.47±2.09	49.63±2.1
HMLN comp 8	-0.18±0.04	0.78	60.32±1.30	48.62±1.2
HMLN comp 9	-0.19±0.03	0.73	61.05±2.06	49.03±1.9
HMLN comp 10	-0.16±0.06	0.80	61.11±2.28	48.17±1.7

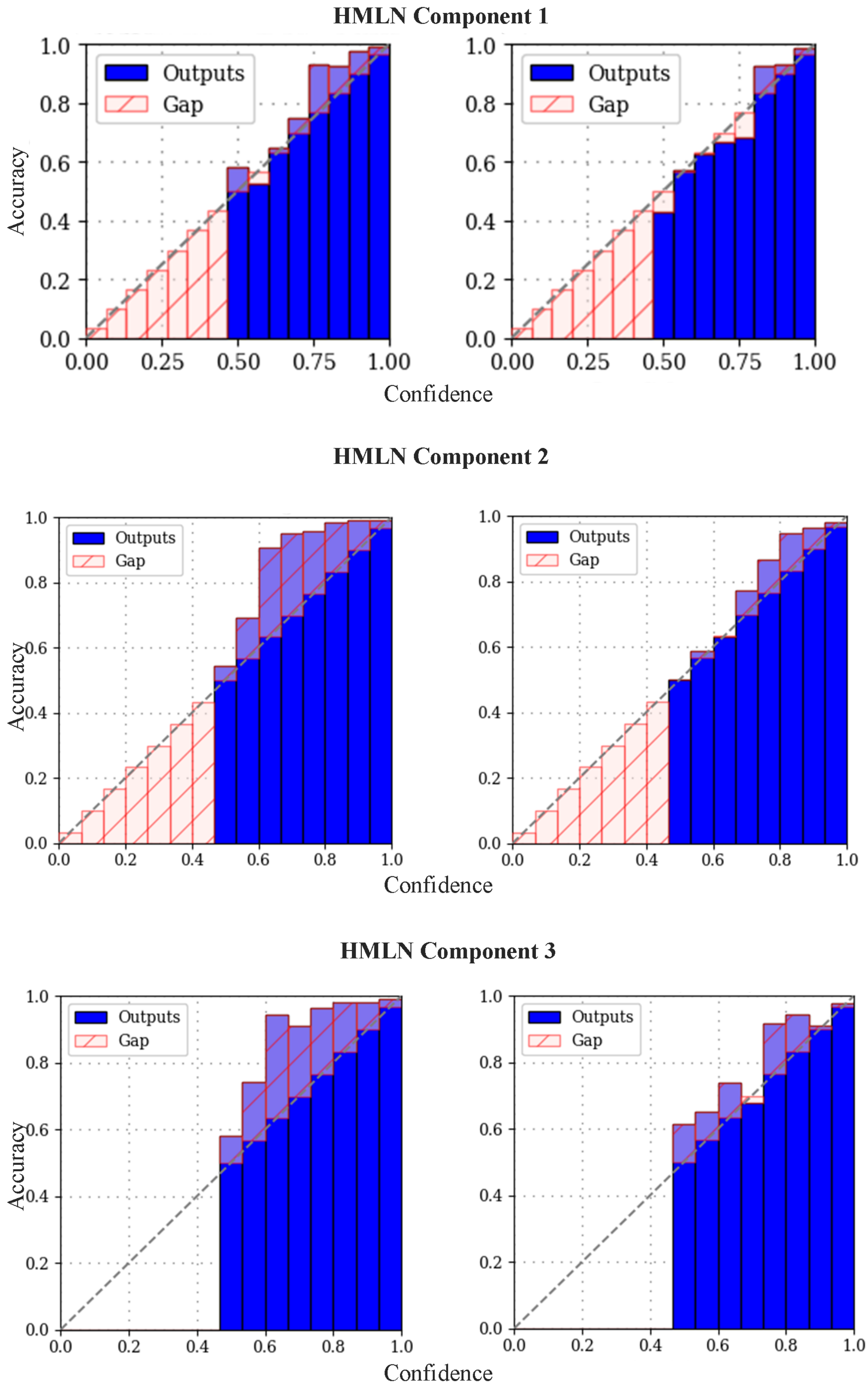


Figure 2: Reliability diagram for calibrating a few of the HMLN components for MixHMLNs on the covariate shifted CORA dataset. The figures on the left are before temperature scaling and the figures on the right are after temperature scaling.

Algorithm 1 Mixture of HMLNs

Input: HMLN structure $\mathcal{M}$, $\mathcal{D} = (\mathbf{y}, \mathbf{x})$, Representations $\{\Phi^i\}_{i=1}^n$

Output: Mixture Model with HMLN parameters $\{\Theta^i\}_{i=1}^K$

```
Initialize $\alpha_i \dots \alpha_K$
Initialize $\{\Theta^i\}_{i=1}^K$
**while** not converged or $t \leq maxiters$ **do**
    // $E - Step$
    Compute the component weight matrix with weights $\gamma_{ij}$ using Eq. (7)
    // $M - Step$
    **for** $j = 1$ through $K$ **do**
        // *Perform gradient descent*
        Compute the MAP assignment and MAP objective $M_j$ to all non-evidence variables given weights $\Theta^j$
        **for** $\theta_k^j$ **do**
            Compute the expected value of the $k$-th formula in the MAP assignment
            Update $\theta_k^j$ using the gradient in Eq. (9)
        **end for**
    **end for**
    **for** $j = 1$ through $K$ **do**
        Update the mixture coefficients $\alpha_i^{(t)}$
    **end for**
**end while**
```

Table 8: Conditional Log-Likelihood on the covariate shifted Citeseer graph for the individual HMLN components with reparameterization.

Experiments	**Noisy Citeseer**			
	CLL	**Avg. Marginals**	**Accuracy (%)**	**Time (secs)**
HMLN comp 1	-0.29±0.08	0.75	64.31±2.14	47.14±1.6
HMLN comp 2	-0.30±0.06	0.71	62.34±2.23	45.46±1.6
HMLN comp 3	-0.33±0.07	0.72	61.34±2.19	42.18±1.1
HMLN comp 4	-0.28±0.04	0.72	62.13±1.93	48.3±2.1
HMLN comp 5	-0.30±0.06	0.72	61.72±1.39	48.44±3.1
HMLN comp 6	-0.27±0.04	0.74	64.80±1.84	44.24±2.1
HMLN comp 7	-0.31±0.03	0.75	64.49±1.37	45.19±2.1
HMLN comp 8	-0.28±0.03	0.70	60.53±1.55	46.71±3.4
HMLN comp 9	-0.28±0.07	0.70	61.49±1.59	46.04±2.8
HMLN comp 10	-0.33±0.05	0.75	60.16±2.36	46.51±2.7

Algorithm 2 Marginal Inference

Input: evidence $\hat{\mathbf{x}}$, non-evidence $\hat{\mathbf{y}}$, test representation $\hat{\Phi}$, probabilistic classifier $\mathcal{C}$, HMLN parameters $\{\Theta^i\}_{i=1}^{K}$
Output: Marginal probabilities for $\hat{\mathbf{y}}$

```
Initialize $\hat{\mathbf{y}}^{(0)}$ to a random state
while Not converged do
    Select component HMLN $\Theta^j$ to sample with probability $\alpha_j$
    Compute the DR for each grounding using $\mathcal{C}$
    Reparameterize the $j$-th HMLN with the DRs using Eq. (16)
    Gibbs sampling steps
    for $y \in \hat{\mathbf{y}}$ do
        From the reparameterized HMLN, sample a single non-evidence variable $y$ using the Gibbs kernel
        if burn-in complete then
            Update marginal estimates for $\hat{\mathbf{y}}$ using the estimator in Eq. (2)
        end if
    end for
end while
return marginal estimates for $\hat{\mathbf{y}}$
```