Approximate Lifted Inference in MLNs using Neural Embeddings

Anonymous Author(s)
Affiliation
Address
email

Abstract

Lifted inference has become the predominant approach to scaling up inference in Markov Logic Networks (MLNs). The main challenge with lifted inference methods is to find symmetries in the MLN. Though there has been a considerable amount of work over the last several years in computing symmetries for MLNs, exploiting approximate symmetries to its fullest possible extent is a challenging task. In this paper, we present a novel approach for symmetry detection based on a neural embedding for objects in the MLN, which we call Obj2vec. Specifically, Obj2vec embeds objects in a low-dimensional space, where symmetric objects are placed close together in this space. Our approach is particularly useful, since it does not require difficult-to-compute features for detecting symmetries, and more importantly it takes advantage of joint dependencies across multiple domains to detect more complex symmetries than is possible if we treat domains independently. Our experiments on several real-world benchmarks validate our hypothesis, and show that Obj2vec-based lifted inference is accurate, scalable and effective in detecting symmetries.

1 Introduction

Markov Logic Networks (MLNs) [**?**] are well-known statistical relational models with intuitive semantics and powerful expressiveness. Specifically, MLNs represent both relational concepts as well as uncertainty using weighted first-order logic formulas. An MLN is a template for Markov networks, i.e., we can instantiate an MLN with objects from a real-world domain and generate Markov networks which represent probability distribution over relational worlds of the MLN. A key challenge with MLNs, as is now well-known, is to develop scalable inference methods for them. In this regard, the idea of *lifted* inference has been a major advancement, where the idea is to utilize symmetries that exist in the MLN due to its template structure, in order to improve scalability of inference.

However, identifying symmetries in the MLN efficiently and effectively is non-trivial. Specifically, the Markov network representing the MLN distribution is usually very large, and it is often intractable to try to process the MLN directly using its underlying Markov network representation. Therefore, several previously developed lifted inference methods try to process the MLN without creating its *ground* Markov network. For instance, popular exact lifted inference methods such as FOVE [**?**], PTP [**?**] and WKBMC [**?**] develop *lifting* rules which identify symmetries in the MLN directly from the first-order structure of its formulas. Similarly, several lifted approaches have been proposed for approximate inference by utilizing symmetries within MCMC[**? ?**], belief propagation [**? ?**], etc.

Unfortunately, it turns out that exact symmetries required for lifted inference are usually absent in real-world MLNs. Therefore, a relatively new direction has been to use approximate symmetries to perform lifted inference in a scalable manner on real-world MLNs. Broeck and Darwiche [**?**] proposed a boolean factorization approach to approximate binary evidence with evidence that has more symmetries, which makes lifting easier. Venugopal and Gogate [**?**] proposed a pre-processing method that clusters each domain using algorithms such as K-Means, and then replaces a cluster by its cluster-center. However, there are some key issues with these approximate lifting methods. First, clustering methods such as K-Means requires feature-engineering. For example, Venugopal and Gogate use features based on counts of formulas satisfied by the evidence. Such features need to accurately encode symmetry in the MLN. Second, joint dependencies between across domains in the MLN are not fully considered while learning the approximate symmetries. For example, consider an MLN that analyzes restaurant reviews. The likelihood of symmetry between two distinct reviews increases if the reviews are written for restaurants that have symmetry. At the same time, restaurants that are somewhat similar may elicit reviews that have a higher likelihood of symmetries. Our main contribution in this paper is to significantly advance the state-of-the-art in detecting approximate symmetries in MLNs by using neural embeddings. Two key advantages of our approach include, it does not use any hand-coded features, and it takes advantage of joint dependencies across domains to detect symmetries more effectively.

Our approach is based on the premise that objects that have similar contexts in the MLN formulas exhibit symmetries. For example, consider once again, the domain of reviews written for restaurants. If, in the ground formulas of the MLN, whenever a ground formula containing review R_1 is true, and a ground formula containing review R_2 is true, if the other objects in the ground formulas such as restaurants, users, etc. (which forms the context for R_1 and R_2) are symmetric, then it is likely that R_1 and R_2 are also symmetric. To detect such symmetries, we develop a novel model similar to the popular skip-gram models (such as Word2vec) which we call Obj2vec model. Obj2vec encodes the context in which objects appear in a ground formula of the MLN that is satisfied by the observed evidence, as “sentences”, and then generates a low-dimensional embedding for these objects. Objects that have similar contexts will be close to each other in the embedded vector-space, and are therefore considered as approximately symmetric to each other. We reduce the total number of objects in the domains of the MLN by removing a subset of objects whose vectors are close to the vectors of objects that we retain in the MLN. As is the case with previous approaches [**? ?**], since our approach is a pre-processing method on the MLN, it can be applied to scale-up any existing inference algorithm.

We perform experiments on three real-world datasets taken from Alchemy [**?**], namely, Webkb for collective classification, Cora for entity resolution, and protein interaction. Further, to illustrate the value of our approach in detecting symmetries, we use a dataset containing Yelp reviews from restaurants. We show that our approach significantly outperforms existing approaches and is effective in detecting symmetries from context.

2 Background

2.1 Markov Logic Networks

Markov logic networks (MLNs) are template models that define uncertain, relational knowledge as first-order formulas with weights. Weights encode uncertainty in a formula. ∞ weight formulas are hard constraints which should always be true. Similarly formulas with $-\infty$ weights are always false. Thus, MLNs offer a flexible framework to mix hard and soft formulas. In MLNs, we assume that each argument of each predicate in the MLN is typed and can only be assigned to a finite set of constants. By extension, each logical variable in each formula is also typed. Thus, each variable corresponds to a specific domain of objects. A ground atom is an atom where each of its variables have bee substituted by constants from their respective domains. A ground formula is a formula containing only ground atoms.

Given a set of constants that represent the domains of variables in the MLN, an MLN represents a factored probability distribution over the possible worlds, in the form of a Markov network. A world in an MLN is an assignment of 0/1 to all ground atoms of the MLN. Specifically, the distribution is given by,

$$\Pr(\omega) = \frac{1}{Z}\exp\left(\sum_i w_i N_i(\omega)\right) \tag{1}$$

where w_i is the weight of formula f_i, $N_i(\omega)$ is the number of groundings of formula f_i that evaluate to True given a world ω, and Z is the normalization constant.

The two common inference problems over MLNs are (1) given observed evidence, compute the marginal probability distribution of a ground atom in the ground Markov network (2) given observed evidence, compute the most probable assignment to all non-evidence atoms. Both queries are computationally hard, and therefore require approximate inference methods.

2.2 Skip-Gram Models

Skip-gram models are used to learn an embedding over words based on the context in which they appear in the training data (i.e., neighboring words). Word2vec [**?**] is a popular model of this type, where we train a neural network based on pairs of words seen in the training data. Specifically, we use a sliding window of a fixed size, and generate all pairs of words within that sliding window. For each pair of words, we present one word of the pair as input, and the other word as a target. That is, we learn to predict a word based on its context or neighboring words. The hidden layer typically has a much smaller number of dimensions as compared to the input/output layers. Thus, the hidden-layer learns a low-dimensional embedding that is capable of mapping words to their contexts. Typically the hidden-layer output is used as features for other text processing tasks, as opposed to using hand-crafted features.

2.3 Related Work

Lifted inference methods exploiting exact symmetries have been explored in several earlier works in the context of exact inference ([**? ? ? ?**]) as well as approximate inference ([**? ? ?**]). Further, the idea of using approximate symmetries for inference has been used to develop general pre-processing methods such as using boolean factorization to generate a symmetric approximation of evidence [**?**], and clustering domain objects to reduce the MLN size [**?**]. More recently, approximate symmetries have been used in specific inference algorithms, such as MAP inference [**?**], sampling-based inference [**?**], and linear programming [**?**]. Kopp et al. [**?**] used symmetry breaking techniques commonly used in the SAT community and applied it to lifted inference. More recently, Anand et al. [**? ?**]proposed the use of non-count symmetries, and contextual symmetries in graphical models. Specifically, they used graph-based tools to detect automorphism groups, and infer symmetries from such groups. There has also been several previous works that have focussed on learning embeddings for relational data [**?**]. However, to the best of our knowledge, ours is the first work to consider lifted inference using neural embeddings.

3 Lifted Inference using Embeddings

We generate a low-dimensional neural embedding for the MLN objects using a model we call Obj2Vec, where approximately symmetric objects lie close to each other in the embedding. To formalize our approach, we assume that our MLN has n domains Δ_1, $\Delta_2 \ldots \Delta_n$. Let $\mathcal{D}$ be the evidence database presented to the MLN, $f_1 \ldots f_k$ be the MLN formulas, and $\mathtt{R}_1 \ldots \mathtt{R}_p$ be the predicates. Our task is to embed the set of domains Δ_1, $\Delta_2 \ldots \Delta_n$ into a lower-dimensional space, and use similarity of the objects in the embedded space to reduce the size of the domains, and create a new domain-set $\hat{\Delta}_1$, $\hat{\Delta}_2 \ldots \hat{\Delta}_n$, where $|\hat{\Delta}_i| \leq |\Delta_i|$.

3.1 Obj2Vec

Consider a simple formula $\mathtt{R}(x) \wedge \mathtt{S}(x, y)$. Let $\Delta_x = \{X_1, X_2, X_3\}$ and $\Delta_y = \{Y_1, Y_2\}$. Let the evidence database be $\mathtt{R}(X_1)$, $\mathtt{R}(X_2)$, $\mathtt{R}(X_3)$, $\mathtt{S}(X_1, Y_1)$, $\mathtt{S}(X_2, Y_1)$, $\mathtt{S}(X_3, Y_2)$. The objects X_1 and X_2 are symmetrical since they can be exchanged in the groundings in which they appear without changing the MLN as shown in Fig. 1. Specifically, they appear along with similar objects (in this case Y_1) in ground formulas of the MLN. Using this, we can define the context as follows.

Definition 1. *Let $\bar{f}$ be a ground formula satisfied by evidence $\mathcal{D}$. We define the context of any object in $\bar{f}$ as the set of other objects in $\bar{f}$.*

Our idea is to detect symmetries based on the above-defined context. For this, consider the following encoding for the MLN objects, where for every satisfied ground formula, we list the objects

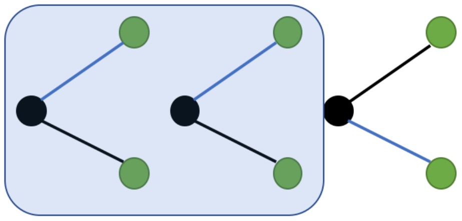

Figure 1: Markov network for $\mathtt{R}(x) \wedge \mathtt{S}(x, y)$. The green nodes correspond to $\mathtt{S}$ and black nodes to $\mathtt{R}$. The blue line indicates that the grounding corresponding to that potential is satisfied by the evidence given by $\mathtt{R}(X_1)$, $\mathtt{R}(X_2)$, $\mathtt{R}(X_3)$, $\mathtt{S}(X_1, Y_1)$, $\mathtt{S}(X_2, Y_1)$, $\mathtt{S}(X_3, Y_2)$. The groundings corresponding to X_1 and X_2 are symmetric.

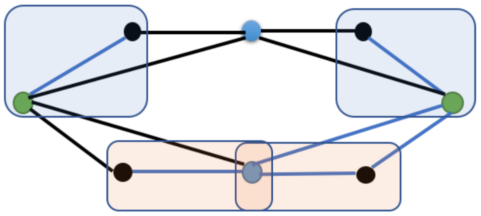

Figure 2: Markov network for $\mathtt{R}(x) \wedge \mathtt{S}(x, y) \wedge \mathtt{T}(y)$. The green nodes correspond to $\mathtt{R}$, black nodes to $\mathtt{S}$ and blue nodes to $\mathtt{T}$. The blue line indicates that the partial grounding corresponding to that potential is satisfied by the evidence given by $\mathtt{R}(X_1)$, $\mathtt{S}(X_1, Y_1)$, $\mathtt{R}(X_2)$, $\mathtt{S}(X_2, Y_1)$, $\mathtt{S}(X_2, Y_2)$, $\mathtt{T}(Y_2)$. Symmetries are shown as shaded regions.

that appear in the context of each other. Therefore, for our previous example, the encoding is, $X_1Y_1; X_2Y_1; X_3Y_2$. Analogous to the input to skip-gram models, we can think of each ground formula as being encoded as a "sentence" in a document, where the sentences are shown to be separated by a ";'. The pair X_1Y_1 encodes that X_1 has a context Y_1 in some grounding of the MLN formula. Notice that X_2 also has a context Y_1.

Now let us consider a neural network architecture that will work on our input encoding. Specifically, the network architecture we use is the same as is used in skip-gram models such as Word2Vec. For Word2vec, the context is defined by a small fixed-size window of words in a document, while for Obj2vec, the context is the objects in a ground formula that is satisfied by the evidence. Therefore, just as in skip-gram models, from each of our encoded sentences X_1Y_1, X_2Y_1 and X_3Y_2, we generate input/output pairs, where we predict an object given another object from its context. That is, given X_1 or X_2 at the input layer, we predict Y_1 at the output layer, and given X_3 as input we predict Y_2 as the output layer. This means that the hidden layer in the model will derive features such that X_1 and X_2 will make common predictions at the output layer. Therefore, the hidden layer representation for both X_1 and X_2 should be similar indicating symmetry of X_1 and X_2. As the context surrounding both X_1 and X_2 becomes more and more similar, the embedding for X_1 and X_2 grows closer to each other. At the same time, since X_3 has a different context, it needs to predict Y_2 at the output layer, and therefore, the hidden layer encoding for X_3 will be different from that of X_1 and X_2.

To generalize the above, let D be the "document" we generate from $\Delta_1, \Delta_2 \ldots \Delta_n$ by encoding the object contexts as sentences. Let $\bar{f}_i^j$ be the j-th grounding of the i-th formula. We now have two cases, $\bar{f}_i^j$ is satisfied by the evidence $\mathcal{D}$, or $\bar{f}_i^j$ is unsatisfied by $\mathcal{D}$. If $\bar{f}_i^j$ is satisfied by $\mathcal{D}$, we encode the the ground objects in $\bar{f}_i^j$ as a sentence in D, indicating the context of the objects in $\bar{f}_i^j$. If $\bar{f}_i^j$ is not satisfied by $\mathcal{D}$, then we do not encode it in D. This is because, the context is not supported by the observed evidence.

Partial Symmetries. One of the consequences of encoding context only based on formulas satisfied by $\mathcal{D}$ is that we lose symmetries of partially satisfied formulas. For example, consider the following MLN, $\mathtt{R}(x) \wedge \mathtt{S}(x, y) \wedge \mathtt{T}(y)$. Let our evidence be $\mathtt{R}(X_1)$, $\mathtt{S}(X_1, Y_1)$, $\mathtt{R}(X_2)$, $\mathtt{S}(X_2, Y_1)$, $\mathtt{S}(X_2, Y_2)$,

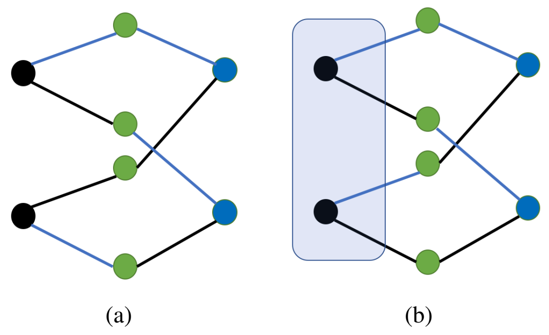

Figure 3: Markov network for $\texttt{R}(x) \wedge \texttt{S}(x, y)$; $\texttt{S}(x, y) \wedge \texttt{T}(y)$. The black nodes correspond to R, green nodes to S and blue nodes to T. The blue line indicates that the grounding corresponding to that potential is satisfied by the evidence. (a) shows the original Markov network. Due to the symmetries when we we consider the pair S and T, we can interchange some of the links between R and S, and detect symmetries on objects corresponding to x as shown in (b).

$\texttt{T}(Y_2)$. The Markov network corresponding to the MLN is shown in Fig. 2. As shown here, there are some partial symmetries, even though the ground formulas are not satisfied by the evidence. Using our aforementioned encoding, since only one ground formula is satisfied by $\mathcal{D}$, our encoding produced is X_2Y_1. Therefore we cannot detect the symmetries that exist between X_1 and X_2 w.r.t the predicates R and S, and the symmetries between Y_1 and Y_2 w.r.t predicates S and T. Therefore, we modify our approach slightly to make it more fine-grained by encoding the context for objects for every pair of predicates in the MLN. The idea is that, for a pair of predicates, if the evidence shows that objects share a context, then it is possible that there are some symmetries between those objects. That is, let $\texttt{R}_i$ and $\texttt{R}_j$ be a predicate pair. For every formula f_i where both $\texttt{R}_i$ and $\texttt{R}_j$ occur, we eliminate all predicates from f_i other than $\texttt{R}_i$ and $\texttt{R}_j$, along with their preceding logical connectives. For example, $\texttt{R}_i(x, y) \wedge \texttt{R}_k(y, z) \Rightarrow \texttt{R}_j(z, x)$ reduces to $\texttt{R}_i(x, y) \Rightarrow \texttt{R}_j(z, x)$. For each grounding of the modified formula, if the evidence $\mathcal{D}$ satisfies the formula, we encode the objects in the grounding as a sentence in our document D. Note that dependencies on self-joined predicates can be naturally handled using this approach by pairing $\texttt{R}_i$ with itself.

Joint Symmetries. One of the advantages of our approach is that symmetries on one object can affect symmetries on another object. For example, consider the formulas $\texttt{R}(x) \wedge \texttt{S}(x, y)$; $\texttt{S}(x, y) \wedge \texttt{T}(y)$. Suppose Y_1 and Y_2 appear in the same context w.r.t S and T in several cases. The embedding for Y_1 and Y_2 will become similar in the hidden-layer of Obj2vec. Now suppose X_1 has a context Y_1, and X_2 has a context Y_2, then, due to the similarity of Y_1 and Y_2, even though X_1 and X_2 have differing contexts, the embedding for X_1 grows closer to X_2. This is illustrated in Fig. 3. Here (a) shows the Markov network for our example MLN given evidence $\texttt{R}(X_1)$, $\texttt{S}(X_1, Y_1)$, $\texttt{R}(X_2)$, $\texttt{S}(X_2, Y_2)$, $\texttt{S}(X_1, Y_1)$, $\texttt{S}(X_1, Y_2)$, $\texttt{T}(Y_1)$, $\texttt{T}(Y_2)$. Due to the symmetries on Y_1 and Y_2 (they both have same context X_1), we can exchange the color of the links between $\texttt{R}(X_2)$, $\texttt{S}(X_2, Y_1)$ and $\texttt{R}(X_2)$, $\texttt{S}(X_2, Y_2)$. This changes the the Markov network as shown in (b), and we can notice symmetries between X_1 and X_2. Thus, in general, we can propagate symmetries across domains, which would help us uncover more complex symmetries in the MLN that are hard to detect when independently clustering the domains.

3.2 Reducing the Objects

We reduce the number of objects in the MLN by reducing the number of symmetric objects. Let $\hat{\Delta}_i \subseteq \Delta_i$, and let $d(X_j, X_k)$ denote the distance between the vectors for X_j and X_k in the embedded vector-space, where X_j, $X_k \in \Delta_i$. We wish to find the subset $\hat{\Delta}_i$ such that the elements in $\hat{\Delta}_i$ are closest to elements in $\hat{\Delta}'_i = \hat{\Delta}_i \setminus \hat{\Delta}_i$ in the embedded vector-space. Given a set A, let $\mathcal{N}(X, A) = \arg\min_{X' \in A} d(X, X')$. We define our objective for Δ_i to be,

$$\arg\min_{\hat{\Delta}_i \subseteq \Delta_i} \sum_{X \in \hat{\Delta}_i} d(X, \mathcal{N}(X, \hat{\Delta}'_i) \tag{2}$$

with the constraint that $|\hat{\Delta}_i| \leq \alpha$. This is required to ensure that our chosen subset of objects is small enough to ensure feasibility of inference. The above problem is a special case of the weighted set-cover problem, and is therefore it is NP-hard to compute the optimal solution. Therefore, we approximate the solution using a greedy approach as follows. We start with an empty $\hat{\Delta}_i$. In each iteration, we sample an object from the pair of objects with closest distance and add it to $\hat{\Delta}_i$. Specifically, in each iteration t, we compute,

$$X^t = \arg\min_{X \in \hat{\Delta}'_i} d(X, \mathcal{N}(X, \hat{\Delta}'_i \setminus X)) \tag{3}$$

We now sample either X^t or $\mathcal{N}(X^t, \hat{\Delta}'_i \setminus X^t)$ with equal probability and add the sampled object, $\bar{X}^t$ to $\hat{\Delta}_i$. We now remove the non-sampled object along with its neighboring object from $\hat{\Delta}'_i$ in a recursive manner. That is, suppose the non-sampled object is $\bar{X}^t_0$, we remove $\bar{X}^t_0$, $\bar{X}^t_1 = \mathcal{N}(\bar{X}^t_0, \hat{\Delta}'_i \setminus \bar{X}^t_0)$, $\bar{X}^t_2 = \mathcal{N}(\bar{X}^t_1, \hat{\Delta}'_i \setminus \bar{X}^t_1)$, and so on. Thus, in each iteration, we remove a cluster of neighbors that are symmetric with the sampled object added to $\hat{\Delta}_i$. We denote the neighborhood of $\bar{X}^t$ by $\mathcal{H}(\bar{X}^t)$. All inference results obtained on $\bar{X}^t$ are assumed to hold for $\mathcal{H}(\bar{X}^t)$.

We stop adding elements to $\hat{\Delta}_i$ when $|\hat{\Delta}_i| \geq \alpha$. However, one problem with this greedy approach is that the solution may get struck in local optima. That is, we may add an element that is close to a few objects, and miss elements that may be symmetrical to many objects even though they may not be the closest pair during an iteration. To avoid local optima, we use a commonly used technique employed in methods such as MaxWalkSAT [?]. Specifically, we intersperse random walks in the solution-space along with the greedy iterations. That is, with probability p, we choose a pair (X, $\mathcal{N}(X, \hat{\Delta}'_i \setminus X)$), where $X \in \hat{\Delta}'_i$ uniformly at random, and sample an object with probability 0.5 from this pair. As before, we remove the non-sampled object and its neighbors recursively from $\hat{\Delta}'_i$. With probability $1 - p$, we choose the pair from which we sample the object greedily.

Given the new domains $\hat{\Delta}_1, \ldots \hat{\Delta}_k$, we change the evidence database $\mathcal{D}$, by removing all evidences that contain objects that are not in the new domains. We map the results obtained from inference using the new evidence database to inference results on the original MLN as follows. The inference results (marginal probability, MAP assignment, etc.) obtained after conditioning on the changed evidence, for an atom grounded with objects $X_1 \ldots X_k$ is assumed to hold for all atoms in the original MLN grounded with objects $\mathcal{H}(X_1) \times \ldots \times \mathcal{H}(X_k)$. Our complete approach is summarized in Algorithm 1.

4 Experiments

4.1 Setup

We conducted our experiments on three benchmarks, Webkb, ER and Protein, all of which are publicly available in Alchemy [?]. Further, we created a new dataset from Yelp reviews, and analyzed our approach with this dataset. We implemented Obj2Vec using the Gensim package [?].

We compared our Obj2Vec-based lifting approach which we refer to as NE, with Venugopal and Gogate's [?] approach (VG) available in Magician [?], where they compress the MLN using K-Means clustering, using features based on counts of formulas satisfied by the evidence. To make the comparison fair, we ensured that we reduced each domain to 10% of its original value in both NE as well as VG. For our Obj2vec architecture, we set the number of number of neurons in the hidden layer as 10% of the total number of objects in the MLN. For a sanity check, we also added an alternative approach, which we refer to as Random, where we sample evidence atoms randomly to reduce the size of the MLN.

4.2 Results

We evaluate our approach on the accuracy of inference, ability to detect symmetries, and scalability.

Algorithm 1: Obj2Vec Lifting

```
Input: MLN structure $\mathcal{M}$, Evidence $\mathcal{D}$, Inference algorithm $\mathcal{I}$, embedding dimensions $m$,
       object bound $\alpha$
Output: Inference results $R$
// Encoding
1  for each predicate pair $(\mathtt{R}_i, \mathtt{R}_j)$ do
2      for each formula $f$ in $\mathcal{M}$ do
3          $f'$ = Eliminate all atoms not corresponding to $(\mathtt{R}_i, \mathtt{R}_j)$
4          Encode groundings of $f'$ satisfied by $\mathcal{D}$ as sentences in document $D$
// Embedding
5  $H$ = Learn a model from $D$ to create a $m$-dimension embedding
// Sampling Objects
6  for each domain do
7      $\Delta_i$ = original domain
8      $\hat{\Delta}_i = \{\}$
9      while $|\hat{\Delta}_i| \leq \alpha$ do
10         $X^t$ = With probability $p$, sample an object from $\Delta_i$ with the closest object-pair
11         $X^t$ = With probability $1-p$, sample an object from $\Delta_i$ from an object-pair chosen
             uniformly at random
12         $\hat{\Delta}_i = X^t \cup \hat{\Delta}_i$ Remove $X^t$ and $\mathcal{H}(X^t)$ from $\Delta_i$
// Inference
13 Convert $\mathcal{D}$ to $\mathcal{D}'$ using new domains $\hat{\Delta}_1, \dots \hat{\Delta}_n$
14 $R'$ = Run $\mathcal{I}$ on $\mathcal{M}$ conditioning on $\mathcal{D}'$
15 $R$ = Project $R$ on original atoms
16 return $R$
```

4.2.1 Accuracy

We first constructed a gold standard for query variables in our benchmarks as follows. For each of the benchmarks, we learned the weights using Magician [?]. We then used around 10% of the benchmark as test data, to ensure that the inference results were obtained were reliable using existing inference methods. We used Magician to perform marginal inference on the benchmark. We then used a threshold to obtain the classification on the query variables of our benchmarks. We consider this result as gold-standard results (GS). We then applied NE and VG to the same test data, and compared the classification of query variables using NE and VG, with GS, in terms of the F1-score. Fig. 4 shows our results. Here, we see that NE significantly outperforms both VG and Random for all three of the benchmarks.

4.2.2 Symmetry Detection

We evaluated the effect of increasing the context on symmetry detection. Specifically, we considered a simple synthetic MLN, where we incrementally add formulas of the form $\mathtt{R}(x) \wedge \mathtt{P}_i(x, y)$. Initially, we have a single formula, and add evidences such that there is a common context for specific objects in Δ_x. Our aim is to detect all those objects of Δ_x as being symmetric to each other. As we add more formulas, the common context for the symmetric objects in Δ_x keeps increasing. Fig. 5 shows our results in terms of the $\%$ of symmetric objects we could detect as we increase the number of formulas, i.e., as we add $\mathtt{R}(x) \wedge \mathtt{P}_1(x, y)$, $\mathtt{R}(x) \wedge \mathtt{P}_2(x, y)$, etc. Our results clearly shows that as the amount of context for symmetric objects in Δ_x increases, symmetry detection becomes more accurate.

Next, we considered the effect of joint dependencies among multiple domains. Specifically, we analyzed the correlation between the symmetry of one domain and the symmetry of other, related domains. For this, we used a real-world dataset consisting of reviews from Yelp. Specifically, we used 7000 reviews written by around 200 users regarding around 800 restaurants. Each review had a score (between 1 and 5) given by a user, and each review was written for a specific restaurant. For our

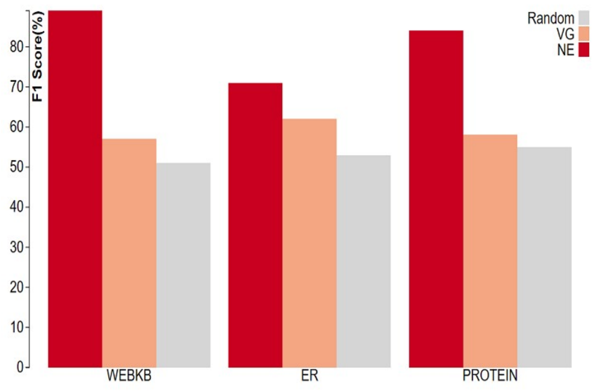

Figure 4: Comparison of F-measures of approximate results on benchmarks using various approaches, as compared to the gold standard (GS) results.

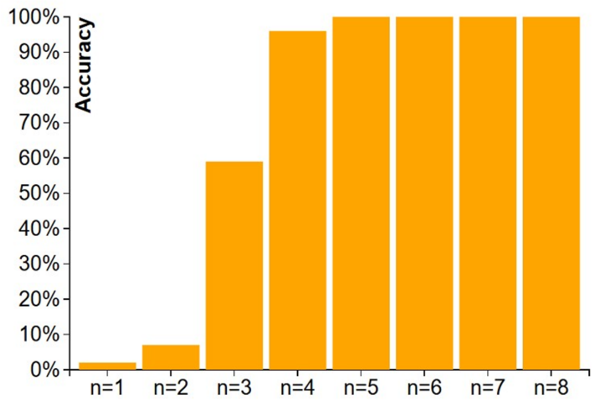

Figure 5: Accuracy in detecting symmetries as we increase the context.

experiment, we considered a simple MLN structure: `Word` $(review,word) \wedge$ `User` $(review,\ user)$ $\wedge$ `Rest` $(review,\ restaurant) \Rightarrow$ `Score` $(review,\ score)$. We embed the domain objects using Obj2vec, and then for each review R, we computed k of its closest neighbors in the embedding. For each of R's neighbors, we computed the similarity between the review vectors, the similarity between user vectors corresponding to users who wrote the reviews, and the similarity between restaurant vectors corresponding to restaurants about which the reviews were written. We then averaged these similarity values over all reviews over varying values of k (between 1 to 10). As shown by the results in Fig 1, symmetries in the reviews, closely correlates to symmetries on the user who wrote the reviews, and symmetries on the restaurants for which the review was written. As our neighborhood size increases, the similarity scores go down uniformly. Thus, each variable's symmetry has a dependency upon the other variables.

4.2.3 Scalability

We evaluate scalability of our approach in terms of time required to generate the embedding and reduce the objects, as well as time for inference on the reduced MLN. Fig. 6 shows our results for time required to generate the reduced MLN. Here, NE takes slightly longer than VG for reducing the MLN on the WebKB and ER datasets, and is almost the same for the protein dataset. The inference time on the reduced MLN was nearly the same for both NE and VG.

Nearest reviews (k)	Avg Review Sim	Avg User Sim	Avg Rest Sim
1	1.00	0.99	1.00
2	0.73	0.99	0.99
3	0.70	0.91	0.89
4	0.60	0.71	0.78
5	0.59	0.71	0.78
6	0.58	0.72	0.78
7	0.57	0.68	0.71
8	0.55	0.72	0.68
9	0.51	0.7	0.72
10	0.50	0.67	0.71

Table 1: Average similarity score of reviews, users and restaurants as we increase number of nearest reviews.

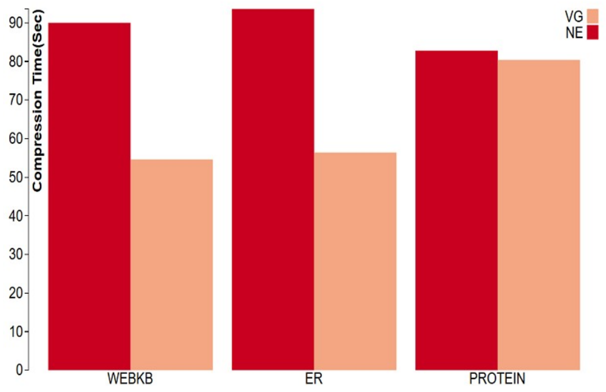

Figure 6: Comparison of running time for NE and VG.

5 Conclusion

In this paper, we presented a novel approach to perform approximate lifted inference in MLNs by leveraging neural embeddings. Specifically, our idea was to exploit the fact that objects with similar contexts (other objects that appear with the object in ground formulas), are likely to be symmetric. Therefore, we encoded the context of an object and trained a neural network on this encoding such that the hidden layer embeds objects in a low-dimensional vector space, where similar objects lie close to each other in this space. We then reduced the number of objects in the MLN by retaining a subset of objects that are close in the embedded-space to the remaining objects that were eliminated from the MLN. Our experiments on real-world datasets showed that our approach is efficient and accurate.

Future work includes leveraging neural models to perform weight and structure learning in MLNs.