Augmenting Deep Learning with Relational Knowledge from Markov Logic Networks

1st Mohammad Maminur Islam
University of Memphis
mislam3@memphis.edu

2nd Somdeb Sarkhel
Adobe Research
sarkhel@adobe.com

3rd Deepak Venugopal
University of Memphis
dvngopal@memphis.edu

***Abstract*—Neuro-symbolic learning, where deep networks are combined with symbolic knowledge can help regularize the model and control overfitting. In particular, for applications where data instances are not independent, domain knowledge can be used to specify relational dependencies which may be hard to infer purely from the data. Symbolic AI models such as Markov Logic networks (MLNs) which are based on first-order logic are designed to represent and reason with uncertain background knowledge. However learning and inference algorithms in such models is known to be slow and inaccurate. In this paper, we develop a novel model that combines the best of both worlds, namely, the scalable learning capabilities of DNNs and symbolic knowledge specified in MLNs. To do this, we infer symmetries in the data based on the relational knowledge encoded in an MLN knowledge base and train a Convolutional Neural Network (CNN) to learn kernels combining symmetrical variables. However, by doing this, we are forced to split the relational data into independent instances for CNN training which may result is a loss of relational dependencies adding noise/uncertainty to the learned model. Therefore, instead of a single model, we learn a distribution over the model parameters. Our experiments illustrate that our model outperforms purely-MLN or purely-DNN based models in several different problem domains.**

***Index Terms*—Markov Logic Networks, Symbolic Models, Relational Learning, Deep Networks**

I. Introduction

Deep Neural Networks (DNNs) have had tremendous successes in challenging domains such as computer vision and natural language processing. Typically, DNNs learn latent representations directly from data and do not incorporate external domain knowledge. However, in many cases domain knowledge greatly helps in learning more generalizable models. Specifically, consider the case of relational data, i.e., where instances have dependencies with other instances in the dataset. In this case, it may be hard to infer all relationships from limited training data. For example, in social networks, a common relational dependency is transitivity, i.e., $\texttt{Friends}(x, y) \wedge \texttt{Friends}(y, z) \Rightarrow \texttt{Friends}(z, x)$. To infer this dependency directly a DNN may need a large number of training examples that supports this relational dependency. Thus, in general, purely relying on data may mean that we ignore important relational dependencies in the learned model. This may result in the DNN overfitting the data (particularly if the dataset is limited) resulting in poor generalization. Regularizing a DNN with domain knowledge helps us learn a more generalizable model more efficiently when the data alone cannot completely explain all the dependencies. In fact, for specialized tasks such as object tracking, it has been shown that external domain knowledge helps DNNs learn accurate models even with very limited datasets [1].

In contrast to DNNs, models such as Markov Logic Networks (MLNs) [2] that are based on symbolic AI are specifically focused on explicitly representing and reasoning with uncertain background knowledge. MLNs in particular represent domain knowledge using first-order logic (FOL) formulas and capture uncertainty by parameterizing the formulas with weights. However, even though tremendous progress has been made in learning and inference for MLNs and other statistical relational models [3] over the last several years, in real-world applications, they still lack accuracy and scalability when compared to deep learning models. For instance, as observed by Khot et al. [4] in the task of question answering, simpler non-relational machine learning models that ignore the relational structure of the problem perform far better than MLNs. One of the problems is that MLNs share a single weight across groundings of a first-order formulas and this sometimes leads to oversimplified models. Considering our earlier social network example, we attach a single weight to the transitive formula $\texttt{Friends}(x, y) \wedge \texttt{Friends}(y, z) \Rightarrow \texttt{Friends}(x, z)$. Clearly, an MLN parameterized by a single value is not rich-enough to accurately answer a meaningful probabilistic query (e.g. what is the probability of $Alice$ and Bob being friends). While it is possible to define a unique weight for each instantiation (using the "+" semantics) in an MLN, learning with thousands of such formulas quickly becomes infeasible. Other models comparable to MLNs such as Probabilistic Soft Logic [5] allow functions instead of weights but have similar scalability issues when the underlying probabilistic model becomes large. In contrast, deep learning methods can fit complex functions to the data in a much more scalable manner. Therefore, to get the best of both worlds, in this paper, we develop a novel Convolutional Neural Network (CNN) based neuro-symbolic model where we use domain knowledge specified in MLN formulas to learn the CNN

This research was sponsored by the National Science Foundation under the awards The Learner Data Institute (award #1934745) and NSF IIS award # 2008812. The opinions, findings, and results are solely the authors' and do not reflect those of the funding agencies.

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

model.

While adding relational knowledge to deep learners have been explored previously, they are typically restricted to adding priors to the DNN [6] or regularizing the loss function with constraints [1]. However, these approaches do not allow us to incorporate information from complex symbolic knowledge bases into DNN learning. Our main contribution in this paper is a novel, flexible approach where MLNs add rich, task-specific background knowledge into the CNN model. The key idea behind our model is to train the CNN based on symmetries in the MLN. For example, in our aforementioned social network example, suppose $Alice$ and Bob have symmetrical social network structures, then the CNN can learn to predict $Alice$'s friendship relations based on Bob's friendship relations. To do this, we learn an embedding for objects in the MLN's domain such that symmetrical objects are placed close to each other in the embedding [7]. We then learn a CNN where each kernel function in the CNN combines object vectors that are approximately symmetrical to each other. However, note that to learn the CNN, we need multiple independent instances while MLN data is relational which means that we have a single long interconnected instance. Therefore, in learning the CNN, we implicitly make independence assumptions in the relational data which can increase the noise or uncertainty in the learned model. To reduce this uncertainty, instead of learning a single parameterization, we learn a distribution over the model parameters using a Bayesian CNN [8] framework.

We evaluate our approach in several challenging real-world problems including, fake review classification, review rating classification, collective classification of webpages and image segmentation. We compare our approach with both state-of-the-art MLN learning methods and deep learning methods where we do not specify background knowledge. Our results show the accuracy and scalability of our approach clearly validating that combining the strengths of MLNs (for background knowledge) and DNNs (for learning) outperforms both purely-MLN and purely-DNN based models in several varied problems.

II. Related Work

While work on combining neural and symbolic learning has a rich histrory [9]–[11], due to advances in deep learning, there has been renewed effort along this direction under the umbrella of neuro-symbolic learning [12] . Rocktaschel and Riedel [13] developed sub-symbolic representations for logical inference operators. Specifically, they developed vector representations for logical symbols and used them for theorem proving in logic. Cohen [14] proposed TensorLog, a deductive reasoning approach and Serafin and Garcez [15] proposed logic tensor networks, an architecture for logical reasoning with deep networks. Our work is also related to knowledge graph embedding and link prediction methods, which are more specialized prediction tasks than the ones we consider in this work. Bordes et al. [16] proposed a neural network architecture to embed knowledge graphs into low dimension structural embeddings. Several approaches for link prediction have been proposed using tensor factorization methods such as ReScal [17], TransE [16] and ComplEx [18]. The idea here is to use factorization methods to predict links in knowledge graphs. More recently Kazemi and Poole [19] proposed a tensor factorization approach called SimpleIE for link prediction that learns embeddings and also allows us to inject background knowledge. In terms of more recent neuro-symbolic learning methods, Manhaeve et al. [20] recently proposed DeepProbLog, that combines a neural network with probabilistic logic. The idea here is to extend probabilistic logic to handle neural predicates, i.e., outputs of the neural network are encoded as a predicate. More recently Xu et al. [21] proposed a semantic loss function to learn deep models with symbolic knowledge. The idea is to encode the constraint specified from logical rules within a differentiable loss function for deep learners. Our approach is orthogonal to this approach and we can use a modified loss function in conjunction with our approach. Recently, Zhang et al. [22] proposed to couple MLNs with GNNs to incorporate knowledge from logic rules. In [7], an approach was proposed to to scale up inference algorithms using neural embeddings, but unlike the approach proposed here, it is not used to learn a model. Our approach is also closely connected to deep symmetry nets proposed by Gens and Domingos [23], where they used a CNN that learns features over symmetry groups capturing more broad invariances in object recognition tasks in computer vision. However, our learning approach is more general since it can learn CNNs that exploit symmetries for a given MLN structure which can encode complex knowledge.

III. Background

A. First-order Logic.

The language of first-order logic (cf. [24]) consists of quantifiers ($\forall$ and $\exists$), logical variables, constants, predicates, and logical connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, and $\Leftrightarrow$). We denote logical variables by lower case letters (e.g., x, y, z) and constants, which model objects in the real-world domain, by strings that begin with an uppercase letter (e.g., A, Ana, Bob). A predicate is a relation that takes a specific number of arguments as input and outputs either TRUE (synonymous with 1) or FALSE (synonymous with 0). The arity of a predicate is the number of its arguments. We assume that each logical variable x has a finite domain of objects Δ_x that it can be substituted with (Herbrand semantics). A ground atom is a predicate where all its variables have been substituted by a constant (we use the terms constants and objects interchangeably) from its domain. For example, $\texttt{Friends}(Alice, Bob)$ is a ground atom obtained by substituting the variables in $\texttt{Friends}(x, y)$.

A first-order formula connects predicates using logical connectives. For example, $\texttt{Friends}(x, y) \Rightarrow \texttt{Friends}(y, x)$. A grounding of a first order formula is one where all variables in the formula have been substituted by constants. For example, $\texttt{Friends}(Alice, Bob) \Rightarrow \texttt{Friends}(Bob, Alice)$. Note that a ground formula evaluates to either True or False. A knowledge-base is a set of first-order formulas.

B. Markov Logic Networks

Markov logic networks (MLNs) soften first-order logic using weights attached to each first-order formula. Each formula in an MLN has a real-valued weight which is shared across all groundings of that formula. For example, $\texttt{Friends}(x, y) \Rightarrow \texttt{Friends}(y, x)$ w. Weights are typically learned by maximizing the likelihood of a relational database [2]. The MLN represents a probability distribution over possible worlds, where a world is an assignment (of either 1 or 0) to every ground atom in the MLN. 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.

C. Bayesian CNN

Regular CNNs are expressive models but prone to overfitting. Bayesian CNNs [8], are more robust, yield uncertainty estimates and can work well even with limited-sized datasets. Specifically, in Bayesian CNNs, we learn a distribution over the convolutional kernels. Inferring the posterior distribution in a Bayesian CNN is a computationally hard problem. The popular approach to learn a Bayesian CNN is to use variational inference. That is, we assume a simple distribution family and learn parameters that minimize the KL-divergence between the approximate distribution and the true distribution. It can be shown that using dropout training, we can perform variational approximation [8]. Further, to make predictions, we can estimate the probability of a query using Monte-Carlo estimates from the learned CNN.

D. Skip-Gram Models

Skip-gram models are used to learn an embedding typically over words based on the context in which they appear in the training data (i.e., neighboring words). Word2vec [25] 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 inputs and output words are encoded as one-hot vectors. 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. Word2vec models can typically scale up to very large text corpora, and can take advantage of pre-training.

IV. Learning Model

There are several strategies we can use to incorporate domain knowledge to the deep learner. One approach that is used quite often is to modify the loss function with constraints that encode some prior knowledge. E.g. Xu et al. [21] proposed a semantic loss function, Russel and Ermon [1] add physics constraints into the loss function for object tracking, etc. However, modifying the loss function for encoding complex types of knowledge is not intuitive. Therefore, we use a more general-purpose language, namely, MLNs using which we can specify even complex domain knowledge quite succinctly. Further, our approach also makes it relatively easy for a human expert to both encode his/her expertise and also to interpret the knowledge base.

Our main hypothesis is that to transfer domain knowledge to the deep learner, we train the deep model based on symmetries encoded in the MLN model. For example, suppose we wish to predict if an individual say Alice needs to receive a treatment, if through our domain expertise, we can find a cohort of individuals similar to Alice, then their treatments can be used to predict the treatment for Alice. In other words, we bias the deep learner to learn from individuals similar to Alice. Further, suppose we learn to make predictions within one cohort, we can apply the same model to other cohorts. Of course, a deep learner can try to and may very well succeed in inferring such symmetries directly from data without any other information. However, given the sparsity of labeled data, explicitly encoding this symmetry information to the deep learner is likely to make learning more efficient. Naturally, one can use the same symmetries to directly learn a parameterization for the MLN. However, as the size of the data grows the learned distribution tends to be highly skewed [26] which is ineffective in making accurate predictions. The main problem is that a single weight in an MLN formula is learned for all groundings of that formula which makes it hard to learn a complex model. We exploit the expressiveness of deep models to learn more accurate parameterizations for relational data.

A. Bayesian Learning Formulation

Let $(X_1, y_1) \ldots (X_n, y_n)$ represent the relational training data where X_i is an atom and y_i is its assignment. As with MLN learning in general, we assume a closed world which means in the training data we have an assignment to every atom in the MLN. We want to estimate a function f that can generate a world (assignment to all atoms). That is, $\mathbf{Y} = f(\mathbf{X})$ where $\mathbf{X}$ represents the atoms and $\mathbf{Y}$ its assignments. We follow the Bayesian approach where $p(f)$ is a prior distribution over the possible functions that can be used for this prediction. The likelihood of predicting the world $\mathbf{Y}$ given that f is the generating function is $p(\mathbf{Y}|\mathbf{X}, f)$. If we can compute the posterior probability $p(f|\mathbf{X}, \mathbf{Y})$, then we can integrate over all functions (f^*) to arrive at a prediction for the probability of generating a new world $\mathbf{Y}^*$ given a new set of atoms $\mathbf{X}^*$.

$$p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{Y}, \mathbf{X}) = \int p(\mathbf{Y}^*|f^*)p(f^*|\mathbf{X}^*, \mathbf{X}, \mathbf{Y})df^*$$

Further, if θ represents the parameters of the function, we can re-write the integral as,

$$p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{Y}, \mathbf{X}) = \int p(\mathbf{Y}^*|f^*)p(f^*|\mathbf{X}^*, \theta)p(\theta|\mathbf{X}, \mathbf{Y})df^*d\theta \tag{2}$$

Since it is intractable to compute the true integral, a standard approach is to use a variational approximation $q(\theta)$ and find parameters that minimizes $KL(q(\theta)||\theta|\mathbf{X}, \mathbf{Y})$. Minimizing this KL divergence exactly is known to be computationally hard, therefore, we sample parameters from the variational approximation q and obtain an unbiased estimator for the ELBO (Evidence Lower Bound) loss given by minimizing the following equation [8].

$$\sum_{i=1}^{N} \ell(\mathbf{Y}, f(\mathbf{X}, \theta^{(i)})) - KL(q(\theta^{(i)})||p(\theta^{(i)})) \tag{3}$$

where $\theta^{(i)}$ is the i-th sample from $q(\theta)$, and $\ell(\mathbf{Y}, f(\mathbf{X}, \theta^{(i)}))$ is the loss between the output generated by the function f using parameters $\theta^{(i)}$ and the true output.

B. CNN Model

To optimize Eq. (3), we select f to be a function that belongs to the CNN function family and ℓ to be the cross-entropy loss. However, to minimize $\sum_i^N \ell(\mathbf{Y}, f(\mathbf{X}, \theta^{(i)}))$, note that we have a single input and output instance which makes it infeasible to learn f. That is, all atoms in $\mathbf{X}$ are related to each other. Therefore, to make it feasible to learn f, we decompose the loss over the atoms as,

$$\sum_{i=1}^{N} \ell(\mathbf{Y}, f(\mathbf{X}, \theta^{(i)})) \approx \sum_{i=1}^{N} \ell(Y_i, f(X_i, \theta^{(i)}))$$

where $\theta^{(i)}$ is sampled from the variational approximation $q(\theta)$ (we expand on this in the next section). However, by considering a single atom X_i to predict Y_i, the CNN function f will not learn to relate different atoms which is essential in relational learning. Therefore, we instead predict an assignment to an atom from a subset of other atoms. Note that this formulation is very similar to the pseudo log-likelihood learning (PLL) formulation for MLNs [2]. The decomposed loss function is given by,

$$\sum_{i=1}^{N} \ell(Y_i, f(\mathbf{X}_i, \theta^{(i)})) \tag{4}$$

where $\mathbf{X}_i \subseteq \mathbf{X}$ that is chosen to predict the assignment to X_i.

Given an atom X_i for which we want to predict its assignment, we determine the optimal subset of atoms $\mathbf{X}_i$ from which the CNN makes its prediction. To do this, we analyze the performance of our model using results in [27]. Specifically, suppose $\hat{G}_N(\mathbf{f})$ represents the empirical Gaussian complexity of function class $\mathbf{f}$, we want to choose the training examples to minimize this complexity. In our case, we choose $\mathbf{f}$ to be the class of two-layer convolutional neural networks with one convolutional layer and one fully-connected layer. Here, $\hat{G}_N(\mathbf{f})$ measures the capacity of the function class and is related to its ability to generalize. Smaller values of $\hat{G}_N(\mathbf{f})$ indicate smaller capacity but better generalization when using the function class $\mathbf{f}$. Larger $\hat{G}_N(\mathbf{f})$ means that we can fit any dataset due to large capacity, but we may overfit leading to poor generalization. Li et al. [28] show that to minimize $\hat{G}_N(\mathbf{f})$, we need to select the convolutional kernels judiciously. Specifically, in our case, for each CNN $f \in \mathbf{f}$, we need select a convolution kernel that minimizes

$$\max_{\mathbf{j} \in \mathcal{S}, \mathbf{j}' \in \mathcal{S}} \sqrt{\sum_{i=1}^{N} ||\mathbf{X}_i(\mathbf{j}) - \mathbf{X}_i(\mathbf{j}')||^2} \tag{5}$$

where $\mathbf{X}_1\ \mathbf{X}_2\ \ldots\ \mathbf{X}_N$ are inputs to the CNN, each of which corresponds to a subset of atoms. $\mathcal{S}$ denotes the shape of the convolutional kernel. Specifically, given an input $\mathbf{X}$, the weights ($\mathbf{w}$) are shared across regions specified by $\mathcal{S}$, i.e., $\sum_{\mathbf{j} \in \mathcal{S}} \mathbf{w}_\mathbf{j} \mathbf{X}[\mathbf{j}]$. Thus, each element of $\mathcal{S}$ specifies the index vector in the input where the convolution kernel is applied. Therefore, from Eq. (5), we understand that to improve generalization, we need to maximize the covariance between $(\mathbf{X}_i(\mathbf{j}), \mathbf{X}_i(\mathbf{j}'))$. To do this, each subset $\mathbf{X}_i$ used as input to the CNN must correspond to atoms that are most "similar" to each other, and the CNN learns kernels over these symmetric groups of atoms. We next describe our approach to encode inputs to the CNN based on similarity of atoms.

C. Composing the Inputs

To compose inputs for CNN learning, we need to convert atoms in the MLN to vectors. We do this by learning an embedding for the objects in the MLN using Obj2Vec [7] and then combine object embeddings into atom embeddings.

Object Embedding. Obj2Vec is a distributed representation for objects in the MLN based on symmetries in the MLN structure. Specifically, very similar to skip-gram models for words, the idea here is to train a neural network that predicts one object in the MLN from other objects in its *context*. Given a relational training dataset, i.e., $(X_1, y_1)\ \ldots\ (X_n, y_n)$ which is an assignment to each atom in the MLN, the context of an object is the set of objects that appears with it in ground formulas that are satisfied (having truth value equal to 1) by the assignments to atoms in the training data. For example, suppose we have a satisfied ground formula, $\texttt{Friends}(A, B) \wedge \texttt{Friends}(B, C) \Rightarrow \texttt{Friends}(C, A)$, then B, C are objects in the context of A and similarly, A, B is in the context of C and C, A is in the context of B. The neural network represents each object using a one-hot encoding and then predicts one object from its context. For example, predict A from B and C in the above example. The hidden layer of the neural network learns to represent each object using a dense vector such that objects having similar contexts have similar hidden layer representations which is also called as the embedding

for the object. Two objects are approximately symmetrical (or approximately exchangeable) in the MLN if they have similar embeddings.

Atom Embeddings. Note that Obj2Vec generates object embeddings. However, for Eq. (4), we require atoms as input. Therefore, we construct a matrix of atom embeddings using additive compositions of object embeddings as follows. Suppose our atom has k objects, corresponding to each object o, we sample a set of objects from the embedding-space such that the object is sampled with probability proportional to its distance to o. That is, objects that are close to o are sampled with high probability. We then compose an atom embedding as an additive composition of embeddings corresponding to the sampled objects. Suppose $v_{o_1}, v_{o_2} \ldots v_{o_k}$ are the vectors sampled corresponding to objects within atom X_i, $\sum_j v_{o_j}$ encodes a row in the input matrix for X_i. We can show that,

Proposition 1. *Let $o_1 \ldots o_k$ be objects in the context of $o'_1 \ldots o'_k$ respectively, then*

$$\sum_{j=1}^{k} v_{o_j} \propto \log \prod_{j=1}^{k} P(o'_j|o_j)$$

Proof. Using the softmax objective function of the basic skip-gram model, we have, $P(o'_j|o_j) \propto \exp(v_{o'_j}^{\top} v_{o_j})$. Therefore, we have the following, $\log P(o'_j|o_j) \propto v_{o'_j}^{\top} v_{o_j}$. Thus, v_{o_j} represents a distribution over its context object o'_j. We can now sum over the vectors to obtain,

$$\sum_{j=1}^{k} v_{o_j} \propto \sum_{j=1}^{k} log P(o'_j|o_j) = \log \prod_{i=1}^{k} P(o'_j|o_j)$$

. □

Thus, from the above proposition, the additive composition of the object vectors produces a vector that encodes the product of their context distributions. This means that suppose we compose a vector from atom X_i, this vector encodes an atom whose objects are symmetrical with objects in X_i. Thus, we use this atom as an input to predict the assignment to X_i. To generate the input *embedding-matrix* for atom X_i denoted by $M[X_i]$ from which we predict Y_i, we sample the neighborhood of objects in X_i k times and generate k composed vectors where each vector is a row in $M[X_i]$. The additive composition approach is similar to the ones used by popular knowledge graph embeddings such as TransE [29]. Note that In principle, other non-additive knowledge graph embedding methods can be adopted for composing the embeddings including multiplicative approaches such as Distmult [30] or RESCAL [17].

D. Uncertainty

From Eq. 4, we see that we sample $\theta^{(i)}$ from $q(\theta)$. To do this, we need to choose a suitable variational approximation from which it is easy to sample and optimize the objective specified in Eq. (3). Gal and Gharmani [8] show that if we choose $q(\theta)$ to be a Bernoulli distribution then the dropout objective exactly corresponds to the objective in specified Eq. (3). Thus, by simply adding a dropout layer after a convolutional layer, we can optimize Eq. (3).

Thus, to optimize Eq. 4, we independently sample the kernels that will be active after each convolutional layer. This means that for each input instance, the structure of the CNN changes and we are essentially learning multiple CNNs which are then averaged together for the final model. This is particularly important for relational learning since the decomposed loss function in Eq. (4) splits the relational data which introduces uncertainty into the learned model. By averaging over multiple models, we are learning a distribution over the kernels of the CNN which helps us quantify this uncertainty. For example, if the prediction for the atoms can be performed accurately over the entire distribution of the kernels, this means that the subsets of atoms that we have chosen for predicting each atom have important dependencies which we are able to learn using the kernel functions much like an MLN formula that connects atoms that have a logical connection together.

To make predictions over the learned CNN, since it is infeasible to perform integration over the full distribution of kernels in the CNN, we approximate this using *MC-dropout*. Specifically, as during learning, using the dropout layer after the convolution layer is equivalent to sampling kernel parameters using the Bernoulli variational approximation $q(\theta)$. We sum over the predicted values corresponding to different samples from $q(\theta)$ to get a Monte Carlo estimate of the predicted probability. Specifically, we predict the probability of an atom X_i as $\frac{1}{T}\sum_{t=1}^{T} p(y_t|M[\mathbf{X}_i], \theta^{(t)})$, where $M[\mathbf{X}_i]$ is the embedding matrix for the atom X_i and y_t is the predicted assignment to X_i in iteration t.

E. Exchangeability

Each row in the input matrix $M[\mathbf{X}_i]$ represents an atom that is similar to X_i. The ordering of these rows is not important. This is different from typical CNN applications that process images since in the case of images changing the ordering of rows of pixels results in a completely different image. In our case, we learn the CNN such that it is invariant to exchanges in the input rows. One approach is to present all possible exchanges or permutations of the input rows with the same output and let the CNN learn a function that is invariant to the ordering of the rows. However, this results in a much larger input space since we need to permute the rows in each input. Instead, we use an approach where we use a permutation-invariant, symmetric function to force the CNN to output the same features regardless of the ordering. Specifically, this function aggregates the information along the columns thus removing the spatial information along the columns. In our case, after the convolutional layers, we use the mean of each column as the permutation-invariant function. Note that other similar functions based on aggregate statistics can be used for this function as well [31]. We then connect the output of the permutation-invariant layer to the dense layers of the CNN.

An illustration of our complete model is shown in Fig. 1. Our learning approach is summarized in Algorithm 1.

Algorithm 1: CNN Learning

Input: MLN $\mathcal{M}$, relational training data $\mathbf{E}$
Output: Learned CNN Model
// Compose Inputs for the CNN
Learn Obj2Vec embedding $\mathcal{O}$ for the objects in $\mathcal{M}$
for *Each atom* X_i *with assignment* Y_i *in* $\mathbf{E}$ **do**
  // Compose the Embedding-Matrix
  for *each object* o *in* X_i **do**
    for $t = 1$ *to* k **do**
      $v_1 \ldots v_m$ = Sampled neighborhood of o in $\mathcal{O}$
      Add $\sum_{i=1}^{m} v_i$ as a row in $M[\mathbf{X}_i]$
  C = Train CNN in Fig. 1 with $(M[\mathbf{X}_i], Y_i)$
return C

V. Experiments

A. Setup

We compared our approach (which we refer to as Markov Logic CNN MLCNN) with i) purely MLN-based models that do not use the inference and learning capabilities of DNNs, ii) purely DNN-based models that do not use background knowledge when learning the model and iii) embedding-based models that uses embeddings from the MLN but not the learning approach of MLCNN.

We evaluated our approach with four different tasks which we refer to as *WebKB*, *Yelp*, *Segmentation* and *Movielens*. The WebKB task and dataset is defined in Alchemy [32] where we classify webpages according to a topic. For Yelp, the task is to classify if a review is fake or not and the associated dataset is available in anomaly detection repository [33]. For Segmentation, we use the TU Darmstadt database of images [34] to perform image segmentation into foreground/background pixels. We used the set of images corresponding to side-views of cows. For the Movielens application, we predict movie ratings and the associated dataset is publicly available at [35].

MLN-based Models. For the MLN based methods, we used two well-known learning and inference systems, Tuffy [36] and Magician [37]. In Tuffy, we perform learning using max-likelihood estimation and for inference, we use MaxWalkSat for MAP inference (where instead of a probability, we predict a joint assignment to all non-evidence atoms) and MCSAT for marginal inference. In Magician, the learning is performed using max-likelihood estimation but with approximate counting methods [38] for improved scalability. Marginal Inference in Magician is implemented using Gibbs sampling [39] once again with special approximate counters for improved scalability.

DNN-based Models. For each of our applications, we implemented a task-specific DNN. For the WebKB, Yelp and Movielens applications we implemented a CNN for text classification based on the well-known architecture specified in [40]. Specifically, in our case, we learn word embeddings from the text and for each instance, we learn kernels over the word embeddings to derive feature maps for the classification task. We refer to this as CNN. For the Segmentation task, we used U-Net [41] which is a state-of-the-art CNN-based approach to segment images. U-Net is a CNN architecture that learns feature maps for the images and then expands these feature maps to get a segmentation for the image.

Embedding-based Model. We implemented an approach where we directly use the atom embeddings within a classifier to predict its assignment without using its symmetry neighborhood. Specifically, we add the embeddings for the objects in the atom and use this vector for predicting the assignment to the atom using logistic regression. Thus, unlike MLCNN, this model does not learn higher-level functions over the embeddings using this approach. We refer to this as EC (Embedding-based classifier).

B. MLCNN Implementation

To learn the Obj2Vec embeddings, we used the Gensim [42] implementation of word2vec with the skip-gram model. We implemented the CNN learning using Tensorflow. The parameter settings for our CNN are as follows. We set the filter-size of 5×5, typically, we do not want a very large filter since we want to learn the kernel over highly correlated variables. We used a neighborhood of size 25, i.e., each input to the CNN has 25 rows. The architecture of our CNN is as follows. We have 4 convolutional layers and 4 max-pooling layers with ReLU units, and one fully connected layer with the softmax output. We present results by varying these parameters in the next section. We tried varied dropout probabilities between .1 to .7 and found best results with dropout probability 0.5. Also, typically, word2vec architectures use a dimension of 300, but from our observation, such a large embedding works well for word embeddings since the corpora is quite large. In our case, we would require significantly more data since the number of weights in the CNN would be very large. Therefore, we varied the embedding dimensions between 10 and 50, and set the number of dimensions to 30 after which the performance did not change significantly. We ran all our experiments on a laptop with 8 GB ram and Intel core i-7 processor.

C. MLN Structure

WebKB. The task in Webkb is to predict the topic of a webpage. We have formulas connecting words to a topic predicate (similar to unigram features used in the bag of words model). We also have formulas to encode the rule that linked webpages have same topics. The set of formulas for this are available in Alchemy [32].

Yelp. The task here is to predict if a review has been labeled as fake or not by Yelp. We once again connect words in our vocabulary to the query predicate that specifies if a review is fake. We also add formulas that connect reviews that are written by the same user. Specifically, it encodes the *homophily* property of the form $\texttt{Writes}(u_1, r_1) \wedge \texttt{Fake}(r_1) \wedge \texttt{Writes}(u_1, r_2) \Rightarrow \texttt{Fake}(r_2)$. That is, a user writes reviews of the same type (fake or not fake). We also add formulas that connect the rating for a review to the query predicate and

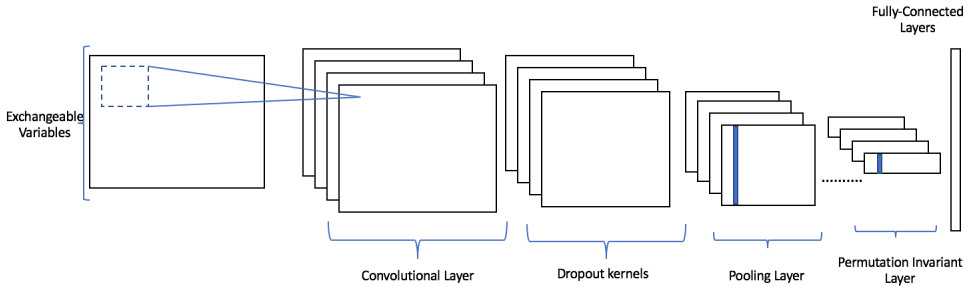


Fig. 1: Illustrating our architecture.

TABLE I: Comparison of MLCNN with other MLN-based models, DNN-based models and embedding-based model (EC). The DNNs for WebKB, Yelp and Movielens is implemented using the CNN text classification approach. The DNN for Segmentation is U-Net, a CNN-based architecture for image segmentation. We show the ROC-AUC scores when the output of the evaluated method is continuous (e.g. marginal inference) and F1-score when the output of the evaluated method is discrete (e.g. MAP inference).

Application	MLCNN	**DNN Models**	**MLN Models**			EC
		CNN/U-Net	Tuffy MCSAT	Magician Gibbs	Tuffy MaxWalkSAT	
WebKB	**92**	73	61	64	54	71
Yelp	**88**	69	56	53	53	61
Segmentation	**97**	90	67	65	67	77
Movielens	**87**	70	46	51	48	76

the restaurant for which the review was written to the query predicate.

Segmentation. The task here is to classify each pixel in an image as a foreground or background pixel. We define regions in which the pixel lies along both the x-axis and the y-axis and add formulas of the form $\texttt{Region}(x_1, y_1, r) \wedge \texttt{Foreground}(x_1, y_1) \wedge \texttt{Region}(x_2, y_2, r) \Rightarrow \texttt{Foreground}(x_2, y_2)$ where (x_1, y_1) and (x_2, y_2) are pixel-coordinates. This rule encodes our knowledge that if two pixels are in the same region, they are likely to be of the same type (foreground/background). We add formulas with a similar structure to connect the average pixel intensities (for R, G and B channels) across 5-pixel neighborhoods along 4 orientations (top, right, bottom, left) to the Foreground predicate. The pixel intensities are discretized to 10 levels since MLNs cannot represent continuous values.

Movielens. The task here is to predict the review rating based on the text in the review. For this task, we used formulas similar to those in the Yelp application, except here, our query atom specifies the rating of a review.

D. Results

We compared the performance of the approaches using five-fold cross validation. We compared classification performance on the query predicates for each dataset. For algorithms that output continuous values (e.g. marginal inference), we report the ROC-AUC score. For discrete outputs (e.g. MAP inference), we present the F1 score. We had a balanced dataset, i.e., for each possible class, the total number of query variables were roughly equal. We summarize the results of our evaluation in Table. I.

WebKB. As shown in Table. I, MLCNN has significantly better accuracy as compared to all the other approaches on all applications. CNN was the next best performer but the significant gap between the performance of CNN and that of MLCNN clearly illustrates the value of incorporating domain knowledge within DNN learning. Further, the poor performance of MLN-based models also clearly show the advantage of leveraging DNN algorithms for relational learning.

Yelp. For the Yelp application, the MLN models do not scale up to the full dataset. Specifically, Tuffy computes a ground Markov network for the MLN and this becomes too large when we consider all the reviews in the dataset. Magician does not ground the full MLN but due to the large number of atoms, the sampling-space is too large for Gibbs sampling to converge. MLCNN and CNN on the other hand can easily process the full dataset which illustrates the scalability of DNN-based learners. Therefore, for Tuffy and Magician, the results are reported for a sub-sampled dataset with 100 reviews. Once again, MLCNN clearly outperforms the other approaches in this task. CNN is again the second-best performer but its performance is significantly behind MLCNN.

Segmentation. For this task, note that we need to classify pixels as foreground/background pixels. In our results, we compute the average accuracy of this classification over 50 images. We can control the number of atoms in this dataset by controlling the number of regions we discretize the image into. Therefore, we were able to run Tuffy and Magician on this dataset. In this task, MLCNN outperforms the other approaches but note that U-Net also yields comparably good performance on this task. For tasks involving natural images, we can think of CNN-based models as implicitly encoding

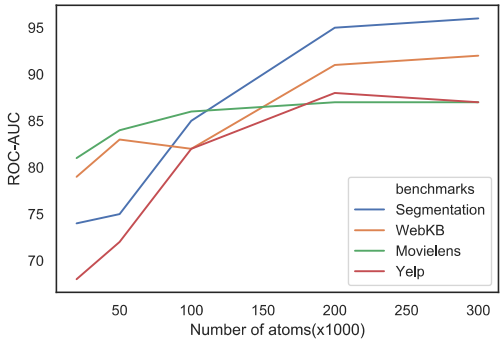


Fig. 2: ROC-AUC scores of MLCNN for varying number of atoms in the datasets.

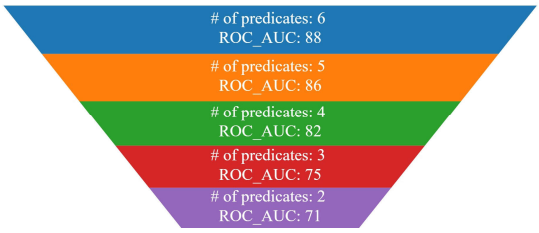


Fig. 3: Illustrating the effect of increasing domain knowledge in the Yelp application. We consider subsets of MLN formulas that contain a subset of predicates and learn MLCNN with these formulas. Results are shown for progressively increasing number of predicates considered in the MLN.

domain knowledge (for e.g., correlations between neighboring pixels) which helps it learn more generalizable models.

Movielens. For the Movielens tasks, just as in the Yelp task, for MLN models, we were only able to run it with 200 reviews since the models could not scale up effectively. MLCNN and CNN on the other hand could easily scale up to process all reviews in this dataset. The accuracy results are very similar to other tasks and the accuracy of MLCNN is again significantly higher than the other approaches.

Scalability. Table II shows the training time for MLCNN as we increase the number of ground atoms in our dataset. We compute the total training time by adding the time required for composing the embedding-matrices for the dataset and the time taken to train the CNN. As we see from the table, the proposed approach scales up well for large datasets, and even when the number of atoms is 300K, we are able to process this within a few minutes. Existing MLN-based learners are incapable of scaling up to such datasets. On the other hand, none of the existing MLN-based methods could scale up to the full Yelp or Movielens datasets. Further, on the Segmentation task, the MLN models failed to scale when the discretized image resulted in more than 50K atoms. Thus, the scalability of MLCNN is significantly better than existing MLN-based relational learning methods.

Fig. 2 shows the accuracy of MLCNN as we increase the number of ground atoms in the data. With increasing number of atoms, the learned kernel parameters can generalize better since the number of symmetry neighborhoods that are considered during training typically increase. This is observed across all our tasks as seen in Fig. 2.

Fig. 3 illustrates the effect of domain knowledge in MLCNN. Specifically, we consider the Yelp dataset (since this has maximum number of predicates). We add MLN formulas incrementally with the constraint that the formula only contains a subset of predicates. We learn MLCNN with increasingly larger subsets of MLN formulas and show the accuracy results in Fig. 3. As we observe in the figure, increasing the number of predicates that are considered in the MLN which implies that we inject more domain knowledge into MLCNN, progressively improves accuracy of our model.

TABLE II: Training time for MLCNN (in seconds) for varying number of ground atoms.

Task	Number of atoms				
	20k	**50k**	**100k**	**200k**	**300k**
WebKB	22	28	35	68	80
Segmentation	45	86	98	114	122
Yelp	56	78	134	195	280
Movielens	32	45	65	80	127

Robustness. One of the advantages of our approach is that we learn a distribution over the CNN kernels which reduces uncertainty in the learned model. Here, we evaluate the robustness of our approach to noisy relations in the dataset. Specifically, we introduce random embedding-matrices into our training dataset to simulate the presence of noisy atoms in the data and evaluate our approach in the presence of noise. Fig. 4 shows our results for varying degrees of noise and dropout. We control these using variables α (to control percentage of noise) and β (to control percentage of dropout). We consider 3 different cases. $\alpha = 1$, $\beta = 0$ indicates the case that we increase noise with the $\%$ indicated by the x-axis but do not use any dropout which is equivalent to learning a single function. $\alpha = 0$, $\beta = 1$ is the case where we do not introduce noise but still learn a distribution over the CNN where the dropout percentage is indicated by the x-axis. Finally, $\alpha = 1$, $\beta = 1$ is the case where we increase noise as indicated by the $\%$ in the x-axis and at the same time increase dropout by the same $\%$. As Fig. 4 shows, as noise increases, the performance of the CNN that learns a single function degrades much faster than the performance of our approach where we learn to combine multiple CNN parameterizations. Further, even when there is no noise, learning a distribution over the CNN parameters does not hurt performance severely as we see from the case where $\alpha = 0$, $\beta = 1$. These results are consistent across all 4 of our benchmarks clearly illustrating the robustness of our approach in the presence of uncertainty.

Exchangeability. We evaluated if the performance of our approach degrades when we use the permutation-invariant

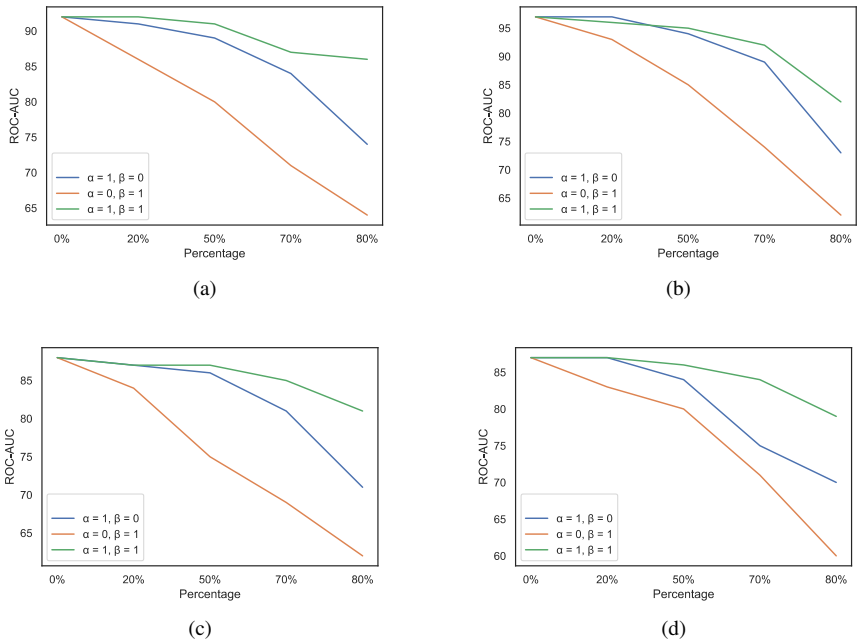


Fig. 4: ROC-AUC scores for MLCNN with varying dropout and noise for (a) WebKB (b) Segmentation (c) Yelp (d) Movielens.

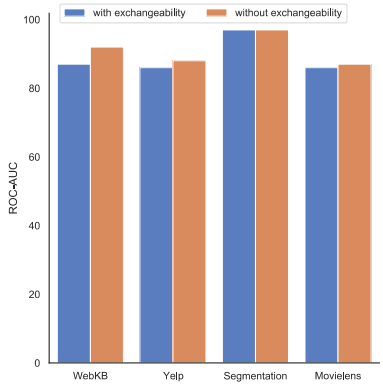


Fig. 5: Illustrating the effect of enforcing permutation invariance in MLCNN.

layer. Specifically, we make the rows in the CNN input order-invariant since each row represents an atom and these atoms can be exchanged with other atoms within the input since they are approximately symmetrical to each other. Fig. 5 shows our results. We can observe from our results over all benchmarks that the performance does not degrade or degrades very slightly when we assume exchangeability of the rows. This result indicates that the atoms that we choose to relate together in the CNN can be truly exchanged which validates the symmetries that we have specified to the CNN.

Hyperparameters. Fig. 6 shows the accuracy of MLNCNN for different hyperparameter settings. Specifically, we show the accuracy results for varying number of convolutional layers, kernel sizes, activation functions and number of kernels. The results are fairly stable across different settings for convolutional layers, kernel sizes and activation functions. For the number of kernels, the optimal setting seemed to be between 4 and 6. At an abstract level, we can think of these as first-order or higher-order formulas since they combine atom embeddings in our model.

VI. Conclusion

We developed a learning approach that trains a CNN based on relational knowledge specified in an MLN and incorporated domain knowledge into the CNN. Specifically, we learn embeddings from the MLN and learn an CNN that combines approximately symmetrical embeddings into higher-order features. To reduce uncertainty in our learned model, we learn a distribution over the parameters of the CNN. Our results show that compared to methods that are based only on MLNs or methods based only on DNNs, our proposed approach has superior performance in varied real-world applications. Future work in this direction includes developing generative learning models using our approach.

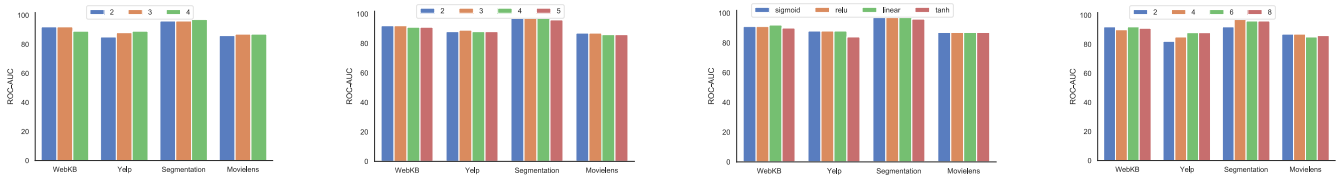
References

[1] R. Stewart and S. Ermon, "Label-free supervision of neural networks with physics and domain knowledge," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press, 2017, pp. 2576–2582.

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

[3] L. Getoor and B. Taskar, Eds., *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[4] T. Khot, N. Balasubramanian, E. Gribkoff, A. Sabharwal, P. Clark, and O. Etzioni, "Exploring markov logic networks for question answering," in *EMNLP*, 2015, pp. 685–694.



(a) (b) (c) (d)

Fig. 6: ROC-AUC scores for varying hyper-parameters in MLCNN (a) Varying number of convolution layers (b) Varying kernel size (figure shows different widths w x w) (c) Varying activation function (d) Varying number of kernels

[5] S. H. Bach, M. Broecheler, B. Huang, and L. Getoor, "Hinge-loss markov random fields and probabilistic soft logic," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 3846–3912, Jan. 2017.

[6] H. Wang and H. Poon, "Deep probabilistic logic: A unifying framework for indirect supervision," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pp. 1891–1902.

[7] M. M. Islam, S. Sarkhel, and D. Venugopal, "On lifted inference using neural embeddings," in *AAAI*, 2019, pp. 7916–7923.

[8] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with Bernoulli approximate variational inference," in *ICLR workshop track*, 2016.

[9] P. Smolensky, "Tensor product variable binding and the representation of symbolic structures in connectionist systems," *Artif. Intell.*, vol. 46, no. 1-2, pp. 159–216, Nov. 1990.

[10] J. W. Shavlik and G. G. Towell, "An approach to combining explanation-based and neural learning algorithms," *Connection Science*, vol. 1, no. 3, pp. 231–253, 1989.

[11] G. G. Towell and J. W. Shavlik, "Knowledge-Based Artificial Neural Networks," *Artificial Intelligence*, vol. 70, pp. 119–165, 1994.

[12] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu, "The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision," in *7th International Conference on Learning Representations, ICLR*, 2019.

[13] T. Rocktäschel and S. Riedel, "End-to-end differentiable proving," in *NIPS*, 2017, pp. 3791–3803.

[14] W. W. Cohen, "Tensorlog: A differentiable deductive database," *CoRR*, vol. abs/1605.06523, 2016. [Online]. Available: http://arxiv.org/abs/1605.06523

[15] L. Serafini and A. S. d'Avila Garcez, "Logic tensor networks: Deep learning and logical reasoning from data and knowledge," in *NeSy@HLAI*, ser. CEUR Workshop Proceedings, vol. 1768, 2016.

[16] A. Bordes, J. Weston, R. Collobert, and Y. Bengio, "Learning structured embeddings of knowledge bases," in *AAAI*, 2011, pp. 301–306.

[17] M. Nickel, V. Tresp, and H. Kriegel, "Factorizing YAGO: scalable machine learning for linked data," in *WWW*, 2012, pp. 271–280.

[18] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *ICML*, 2016, pp. 2071–2080.

[19] S. M. Kazemi and D. Poole, "Simple embedding for link prediction in knowledge graphs," in *Neural Information Processing Systems*, 2018, pp. 4289–4300.

[20] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. D. Raedt, "Deepproblog: Neural probabilistic logic programming," *CoRR*, vol. abs/1805.10872, 2018. [Online]. Available: http://arxiv.org/abs/1805.10872

[21] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. V. den Broeck, "A semantic loss function for deep learning with symbolic knowledge," in *ICML*, 2018, pp. 5498–5507.

[22] Y. Zhang, X. Chen, Y. Yang, A. Ramamurthy, B. Li, Y. Qi, and L. Song, "Efficient probabilistic logic reasoning with graph neural networks," in *8th International Conference on Learning Representations, ICLR*, 2020.

[23] R. Gens and P. M. Domingos, "Deep symmetry networks," in *Neural Information Processing Systems*, 2014, pp. 2537–2545.

[24] M. R. Genesereth and E. Kao, *Introduction to Logic, Second Edition*. Morgan & Claypool Publishers, 2013.

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[26] H. Mittal, A. Bhardwaj, V. Gogate, and P. Singla, "Domain-size aware markov logic networks," in *AISTATS*, 2019, pp. 3216–3224.

[27] P. L. Bartlett and S. Mendelson, "Rademacher and gaussian complexities: Risk bounds and structural results," *J. Mach. Learn. Res.*, vol. 3, pp. 463–482, 2003.

[28] X. Li, F. Li, X. Z. Fern, and R. Raich, "Filter shaping for convolutional neural networks," in *ICLR*, 2017.

[29] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, 2013, pp. 2787–2795.

[30] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *CoRR*, vol. abs/1412.6575, 2014.

[31] J. Chan, V. Perrone, J. Spence, P. Jenkins, S. Mathieson, and Y. Song, "A likelihood-free inference framework for population genetic data using exchangeable neural networks," in *Neural Information Processing Systems*, 2018, pp. 8594–8605.

[32] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, and P. Domingos, "The Alchemy System for Statistical Relational AI," Department of Computer Science and Engineering, University of Washington, Seattle, WA, Tech. Rep., 2006, http://alchemy.cs.washington.edu.

[33] S. Rayana and L. Akoglu, "Yelp Dataset for Anomalous Reviews," Stony Brook University, Tech. Rep., 2015, http://odds.cs.stonybrook.edu.

[34] B. Leibe, A. Leonardis, and B. Schiele, "Combined object categorization and segmentation with an implicit shape model," in *Proceedings of the Workshop on Statistical Learning in Computer Vision*, Prague, Czech Republic, May 2004.

[35] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, 2016.

[36] F. Niu, C. Ré, A. Doan, and J. W. Shavlik, "Tuffy: Scaling up statistical inference in markov logic networks using an rdbms," *PVLDB*, vol. 4, no. 6, pp. 373–384, 2011.

[37] D. Venugopal, S.Sarkhel, and V. Gogate, "Magician: Scalable Inference and Learning in Markov logic using Approximate Symmetries," The University of Memphis, Tech. Rep., 2016, https://github.com/dvngp/CD-Learn.

[38] S. Sarkhel, D. Venugopal, T. A. Pham, P. Singla, and V. Gogate, "Scalable training of markov logic networks using approximate counting," in *AAAI*, 2016, pp. 1067–1073.

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

[40] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014, pp. 1746–1751.

[41] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, vol. 9351, 2015, pp. 234–241.

[42] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010, pp. 45–50.