On Learning from Relational Data using Symmetries

Abstract

Adding background knowledge to deep learning models can help regularize the model and control overfitting. In particular, background knowledge can help in domains that have complex relational dependencies. Symbolic relational models such as Markov Logic networks (MLNs) which are based on first-order logic can easily incorporate uncertain background knowledge. However, performing learning and inference in such models is slow and inaccurate. In this paper, we develop a novel model based on CNNs to perform reasoning in relational problems that is more accurate and scalable than existing methods. The key underlying principle behind our approach is that to incorporate relational knowledge, we use symmetries in the domain to train the CNN. However, in order to train the CNN, we need to split the relational data and thus may lose dependencies. To account for this, instead of a single model, we learn a distribution over the model parameters. Further, we exploit exchangeability among symmetric variables to reduce the input-space and make learning more efficient. We compare our approach to MLN-based relational learning methods as well as relational learners designed for knowledge-base completion tasks and show that our proposed model yields accurate results in several varied problem domains.

1 Introduction

Deep learning has had successes in several domains such as computer vision, natural language processing, game playing, etc. However, deep learning methods generally rely purely on data and very little on domain knowledge which is one of the causes of overfitting. Thus, a fundamental issue in deep learning is to learn representations that encode or exploit domain knowledge. On the other hand, methods that have origins in symbolic AI such as Markov Logic Networks (MLNs) [7] were built specifically for representing and reasoning with uncertain, complex background knowledge using the language of first-order logic (FOL). However, learning and inference methods in these models are well-known to be notoriously slow, inaccurate and not scalable to large problems. Therefore, utilizing the power of deep learning algorithms and incorporating background knowledge into the model we can make progress in relational learning problems. In this paper, we present an approach that takes steps towards this goal by using deep networks to learn representations from symmetrical relationships that can be inferred from symbolic models such as MLNs.

While it is desirable to infuse deep learning with relational knowledge, the barrier is that the fundamental language used by these models are different. Specifically, relational models such as MLNs represent concepts using symbols and formulas, while deep learners work on vector representations. Our hypothesis is that if we want to transfer knowledge encoded in symbols and formulas, we can do so by ensuring that we learn from vectors that encode symmetries in the domain. While it can be argued that we can perform relational inference with these symmetries directly on the MLNs or other symbolic models using techniques such as *lifted inference* methods [21], there are many fundamental problems with this approach. In particular, the MLN parameterization has considerable drawbacks and yields poor accuracy in complex problems. For instance, as observed by Khot et al. [13] 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. The problem is that MLNs share a single weight across ground formulas and this leads to oversimplified models. E.g. Consider a social network application, here, we can encode a transitive formula to represent all friendships in the network such as $\texttt{Friends}(x, y) \wedge \texttt{Friends}(y, z) \Rightarrow \texttt{Friends}(x, z)$ to which we attach a weight. 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 using any MLN learning approach. Other models comparable to MLNs such as Probabilistic Soft Logic [1] allow functions instead of weights but have similar scalability issues when the ground network become large. In contrast, using state-of-the-art deep learning methods will allow us to fit more complex functions to represent relationships in the data in a scalable manner.

Our main contribution in this paper is a Convolutional Neural Network (CNN) based model to learn from relational data that incorporates symmetries derived from formulas of the MLN. Note that learning from relational data is fundamentally different from learning a typical deep model. Relational data consists of a single interconnected instance, while deep models such as CNNs generalize by learning from multiple instances. To perform learning from the relational database, we split the relational data and learn functions over subsets of variables in the data. Importantly, the variables over which we learn functions share symmetries in the MLN structure. However, since we may not be able to represent all dependencies specified in the relational data, this can increase uncertainty in the learned model and therefore, we learn a distribution over the model parameters using variational approximations. In particular, we learn the CNN model with distributions over kernels. The CNN model is trained using vector embeddings for variables of the MLN that are learned based on the MLN structure [10]. Further, we explicitly model exchangeability of symmetric variables in the CNN resulting in a reduced size of the possible input-space.

We apply our proposed approach in several relational problems including fake review detection, review rating prediction, collective classification, knowledge-base completion, etc. We demonstrate the accuracy of our approach as compared to state-of-the-art systems that i) directly use MLN-based relational learning and ii) methods based on neural networks and tensor factorization that perform relational learning but without the help of knowledge or symmetries specified in an external model such as MLNs.

2 Related Work

Combining neural networks with symbolic systems has a rich history of research. Smolensky [27] developed Tensor Product Representations to vectorize symbolic structures using compositions of embeddings. Shavlik and Towell [26, 28] unified propositional logic models unified with neural networks. Several approaches have been proposed recently in this direction. Rocktaschel and Riedel [24] 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 [6] proposed TensorLog, a deductive reasoning approach and Serafin and Garcez [25] 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. [4] 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 [19], TransE [4] and ComplEx [29]. The idea here is to use factorization methods to predict links in knowledge graphs. More recently Kazemi and Poole [12] 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. [16] 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. [32] 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. In [10], an approach to cluster based on object embeddings is proposed to scale up inference algorithms, but unlike our approach, they do not learn a model. Our approach is also closely connected to deep symmetry nets proposed by Gens and Domingos [9], 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.

3 Background

3.1 Markov Logic Networks

Markov logic networks (MLNs) are symbolic models that represent uncertain domain knowledge using the language of first-order logic and probabilistic graphical models. Specifically, a formula in an MLN represents relational dependencies across different variables in the model. Each formula in an MLN has a real-valued weight which is shared across all instantiations of the formula (called groundings of the formula). Weights are typically learned by maximizing the likelihood of a relational database [7] Learning an MLN is computationally hard since computing the gradient requires inference over a large probabilistic graphical model. The MLN with learned weights represents a Markov network, where each ground formula is a potential in the Markov network.

3.2 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.

3.3 Embeddings

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 [17] 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.

Obj2Vec. Obj2Vec [10] is a distributed representation for objects in the MLN based on symmetries. The main idea in this approach is to train a neural network that predicts objects from other objects. Specifically, like skip-gram models, Obj2Vec predicts the *context* of an object, namely other objects in satisfied ground formulas. The hidden layer of the neural network learns to represent one-hot encoded input object vectors in a reduced dimension such that objects that appear in similar contexts are placed close together in the embedded space. To learn the embedding, given an MLN structure, for every formula of the MLN satisfied by the evidence, we predict an object given other objects in its context. In [10], Obj2Vec was used to scale up approximate inference in MLNs by clustering the object embeddings and sampling from each cluster, thus reducing the number of objects and the size of the MLN. Since objects that appear with the same objects repeatedly (i.e., same context) will have embedding vectors that are similar, the clustering produces sets of approximately symmetric objects. Note that this approach was purely used to scale up inference and did not change the learning methods.

4 Learning Model

Our approach trains a deep learner on relational data incorporating domain knowledge. Note that we can specify domain knowledge using several approaches. For example, we can use regularization terms in the deep learner such that the loss function complies with our domain knowledge, we can place priors based on the distribution learned by the model, etc. However, we want to use a more flexible language to specify complex, relational domain knowledge. MLNs have their roots in first-order logic and therefore are well-known to be very expressive and compact since we can specify constraints related to different groundings of a formula using a single first-order structure. Further, it is also easy for a human expert to write down domain knowledge using MLN formulas as compared to changing objective functions for learning, etc. Therefore, here, we choose to use MLN semantics to specify relational dependencies.

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 infer such symmetries directly from data but using background knowledge will make learning easier since we can explicitly tell the deep learner regarding the symmetries. Relational learning methods such as those based on MLNs can also do the same (e.g. lifted inference), however, the disadvantage is that the models learned are not accurate and/or scalable. Specifically, the distribution of the MLN is based on the ground Markov network which can be very large, learning is hard and at the same time, for large domain-sizes, MLNs encode highly-skewed distributions [18] which do not yield accurate results. Further, the parameterization of attaching a single weight for all instantiations of a formula is ineffective when we are dealing with complex problems and adding more weights increases complexity of learning. Therefore, using a deep learning architecture for learning can help us scale up and learn more complex parameterizations.

4.1 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. We want to estimate a function f that can generate a world (assignment to atoms). Since the data is relational, the function may depend upon all the instances jointly, so a general form of this function is, $\mathbf{Y} = f(\mathbf{X})$, where $\mathbf{X}$ represents the input and $\mathbf{Y}$ its assignments. Let $p(f)$ be a prior distribution over the possible functions. The likelihood of generating the world 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 to arrive at a prediction for the probability of generating a new world $\mathbf{Y}^*$ given a new set of inputs $\mathbf{X}^*$.

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

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

$$p(\mathbf{Y}^*|\mathbf{X}^*, \mathbf{Y}, \mathbf{X}) = \int p(\mathbf{Y}^*|f^*)p(f^*|\mathbf{X}^*, \theta)p(\theta|\mathbf{X}, \mathbf{Y})df^*d\theta \quad (1)$$

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 the ELBO loss. In this case, we sample parameters from q and can obtain an unbiased estimator for the ELBO loss as [8],

$$\sum_{i=1}^{N} \ell(\mathbf{Y}, f(\mathbf{X}, \theta^{(i)}) - KL(q(\theta^{(i)})||p(\theta^{(i)})) \quad (2)$$

where $\theta^{(i)}$ is sampled from $q()$, 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.

4.2 CNN Model

We optimize the loss $\ell(\mathbf{Y}, f(\mathbf{X}, \theta))$ where f is parameterized as a neural network. However, since we have a single inter-connected instance as our training data, we cannot directly optimize the loss $\ell(\mathbf{Y}, f(\mathbf{X}, \theta))$. One approach is to decompose the loss by simply predicting each atom independently. Specifically,

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

However, such an approach is an oversimplification and will not learn relationships across different variables. Therefore, we propose a CNN model where we relate together subsets of variables. In general, the number of possible subsets are too large and therefore, it is infeasible to capture all possible relational dependencies. Therefore, the idea is to sample variables in the relational data such that learning from the training instances improves generalization performance. To do this, we analyze the performance of our model using results in [2]. Specifically, suppose $\hat{G}_N(F)$ represents the empirical Gaussian complexity of function class F, we want to choose the training examples to minimize this complexity. In our case, we choose F to be a two-layer convolutional neural network with one convolutional layer and one fully-connected layer. Here, $\hat{G}_N(F)$ measures the capacity of the function class and is related to its ability to generalize. Smaller values of $\hat{G}_N(F)$ indicate smaller capacity but better generalization when using the function class F. Larger $\hat{G}_N(F)$ means that we can fit any dataset due to to large capacity, but we may overfit leading to poor generalization. Li et al. [15] show that to minimize $\hat{G}_N(F)$, where F is a CNN, we need to select the convolutional kernels in F judiciously. Specifically, in our case, we need select a convolution kernel that minimizes

$$\max_{\mathbf{j}\in\mathcal{S},\mathbf{j}'\in\mathcal{S}} \sqrt{\sum_{i=1}^{N} ||\mathbf{X}_i(\mathbf{j}) - \mathbf{X}_i(\mathbf{j}')||^2} \tag{3}$$

where $\mathbf{X}_1$ $\mathbf{X}_2$ ... $\mathbf{X}_N$ are inputs to the CNN, each of which corresponds to a subset of variables. $\mathcal{S}$ 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. (3), 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 $\mathbf{X}_i$ must encode variables that are symmetrical to each other, and the CNN learns kernels over these symmetric groups.

4.3 Composing the Inputs

To compose the inputs, we need to convert variables in the MLN to vectors. We do this by learning an embedding for the objects in the MLN as proposed in [10]. Specifically, we use a skip-gram model to predict an object from an object that appears in its context (same satisfied ground formula). We can think of the embedding as a latent representation for the ground formulas. If two object embeddings are close to each other, this implies that, they are exchangeable in the atoms and formulas of the MLN. For each atom in the training data, X_i, we compute its input matrix $\mathbf{X}_i$ using k atoms that are in the vicinity of X_i in the embedding. Specifically, note that one of the properties of skip-gram based embeddings is their linear structure. That is, we can create embeddings for more complex constructs as an additive composition. In our case, suppose our atom has k objects, we sample from the neighborhoods of each of the objects with a probability proportional to their distance and compose them to create a new atom embedding. Suppose v_{o_1}, v_{o_2} ... v_{o_k} are the vectors sampled for atom X_i, we add the vectors $\sum_j v_{o_j}$ to obtain a row in the input matrix for X_i. We can show that,

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

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

. $\square$

Thus, the additive composition of the object vectors produces a vector that encodes the product of their context distributions. Specifically, suppose we generate a vector additively from objects sampled from neighborhoods of objects in atom X_i, the resulting vector encodes a virtual atom whose objects are exchangeable with objects in X_i. Thus, each row in the CNN input matrix corresponding to X_i encodes an atom exchangeable with X_i. The additive composition approach is similar to the ones used by popular knowledge graph embeddings such as TransE [3]. Specifically, the idea in TransE is that we can represent relation and entity embeddings using a translation model. That is, given the triplet (h, r, t), where h and t are entities and r is a relation, TransE denotes the relation as a translation in the embedding space, i.e., the embedding of the tail entity t is close to the embedding of the head entity h translated by the relation embedding. Note that In principle, other non-additive knowledge graph embedding methods can be adopted for composing the embeddings including multiplicative approaches such as Distmult [33] or RESCAL [19].

4.4 Uncertainty

Learning from relational data using a CNN adds uncertainty to the model. Specifically, recall that in MLNs, we learn weights that optimize the likelihood from a single instance. However, to learn using CNNs, we split the relational instance based on symmetries. This means that we do not consider all relational dependencies jointly when we learn the model and only consider the relationships between atoms that are most symmetric to each other. However, learning a single model in this case means that the same kernels can learn to relate symmetries across all variables. However, different subsets of variables may be related through different symmetry structures. Therefore, to account for this uncertainty, instead of a single model, we learn a distribution over the CNN kernels. Specifically, using the variational formulation in Eq. (2), we approximately integrate over the kernel functions using Bernoulli variational distributions. It turns out that using dropout layers after convolutional layers exactly corresponds to approximating the objective in Eq. (2) where the variational approximation $q()$ is a Bernoulli distribution [8]. Specifically, we sample Bernoulli variables and based on the sampled variables, the kernel outputs are retained or turned off (set to 0) after performing the convolutional operation. Thus, after each convolutional layer, we effectively dropout a subset of nodes in the next layer before using the pooling layer.

4.5 Exchangeability

The rows in each input to the CNN are exchangeable. That is, the order in which the rows occur are not important. Analogously, suppose each dimension in the embedding represents a latent formula, then each variable (corresponding to a row) in the input can be exchanged with other variables since we assume that the variables are symmetric in the MLN formulas. To make learning more efficient, we encode this exchangeability within the CNN. 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 approximately 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 [5]. We then connect the output of the permutation-invariant layer to the dense layers of the CNN.

4.6 Predictions

To make predictions, we need to integrate over the parameter distribution. We use *MC-dropout* to approximate this where the dropout layer is equivalent to sampling kernel parameters using the variational approximation $q()$. We then sum over the predicted values to get a Monte Carlo estimate of the predicted probability. Specifically, we predict the probability of the output as $\frac{1}{T}$ $\sum_{t=1}^{T} p(y_t|\mathbf{X}_i, \theta^{(t)})$, where $\mathbf{X}_i$ is the input matrix and y_t is the predicted value in iteration t. An illustration of our complete model is shown in Fig. 1.

5 Experiments

5.1 Setup

We compared our approach (which we refer to as CNN) with the following approaches that perform learning and inference using MLNs. Tuffy [20] is a state-of-the-art MLN system that performs weight learning using MaxWalkSat (a MAP inference algorithm) to approximate the gradient during the optimization process. Magician [30] performs weight learning using Gibbs sampling to approximate the gradient and also uses special purpose approximate counters to scale up to Gibbs sampling over large datasets. We added another approach where we directly use the embedding of an atom for classifying its state without using its symmetry neighborhood. Thus, the CNN does not take advantage of correlated atoms using this approach. We refer to this as *EC* (Embedding-based classifier). Further, we also compared our approach with other relational learners (TransE [3], Rescal [19], etc.) that are not MLN-based for a knowledge-base completion task.

To learn the embeddings, we used the Gensim [23] implementation of word2vec with the skip-gram model. We implemented the CNN learning using Tensorflow. 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. For the dropout layer, we set the probability as 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.

Datasets We evaluated our approach on several datasets. Webkb from Alchemy [14] where we need to classify webpages according to a topic. Yelp Reviews dataset [22] where we classify if a review is fake or not fake. The TU Darmstadt Database of Images where we perform image segmentation into foreground/background. The Movielens dataset where we predict movie ratings. The Freebase dataset where we perform knowledge-graph completion.

Domain Knowledge. For each task, we encoded MLN formulas as domain knowledge. The task in the Webkb dataset is to predict the topic of a webpage. Words in the webpage are connected to the topic predicate and we have a formula that connects webpages that are linked together to the topic predicate. For the reviews dataset, the task is to predict if a review has been labeled as fake or not by Yelp. We again connect words in the review to the query predicate and we also add formulas that connect reviews that are written by the same user. Specifically,

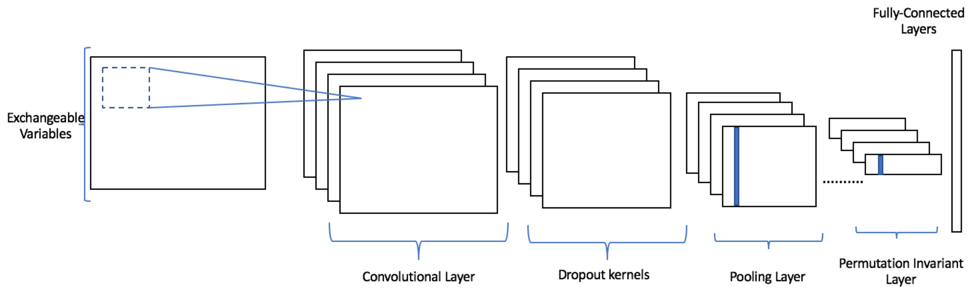


Figure 1: Illustrating our architecture.

	cnn(roc-auc)	**tuffy-mcsat(roc-auc)**	**magician-gibbs(roc-auc)**	**Tuffy-map(F1)**	**EC(roc-auc)**
webkb	92	61	64	54	71
reviews	88	56	53	53	61
images	97				77
movielens	87	46	51	48	76

Table 1: Comparison of different approaches on different datasets.

it encodes the *homophily* property where a user writes reviews of the same type (fake or not fake). We also connect the rating for a review to the query predicate and the restaurant for which the review was written to the query predicate. For the images dataset, the task is to classify each pixel in an image as a foreground or background pixel. For this, we defined a finite domain of regions in which a pixel lies along the x and y axis. We then defined a relation between the pixel in a specific region, and the average intensity of its neighborhood. We considered the average intensity of 5 pixels at the top, bottom, left and right of the pixel under consideration. We quantized the intensities to 10 levels. We connected each of these relations to the query predicate. We experimented with the side-views of 50 cow images from the dataset. For the Movielens dataset, we used formulas similar to those in the Yelp reviews dataset, except here our query variables classify the rating of a review. Since the task on the Freebase dataset is slightly different, we describe it in the final subsection.

5.2 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 perform marginal inference and for our CNN which outputs probabilities, we report the ROC-AUC score. For MAP inference, since it outputs the MAP state, 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 show the results in Table. 1.

As shown in Table. 1, our approach has significantly better accuracy on WebKB as compared to all the other methods. To learn MLN weights for Tuffy and Magician, we used SVM weights to set the weights for the high-dimensional formulas (word-based formulas). We then used the Tuffy and Magician algorithms to learn weights on the remaining formulas. As it can be seen from the results, using standard MLN parameterization, the classification accuracy is quite poor. The poor scores are a result of both simple parameterization as well as performance of algorithms that use approximate inference for setting the weights which may not always learn an accurate model. In comparison, our approach (CNN) yields much higher accuracy. Further, comparing with the accuracy of EC shows the power of taking advantage of CNN capabilities to model the correlations in the dataset.

The Reviews dataset presents a harder challenge since it is difficult to understand why a review is labeled as fake by Yelp. In fact, even human experts have a hard time distinguishing fake from real reviews [11]. For this dataset, on Tuffy and Magician, we could not use the full database since it was too large. Therefore, we randomly sampled 100 reviews, generated a smaller database and present results based on that database. Similar to the above approach, we learned the weights using a combination of SVMs and MLN learning. Note that with such a small database, our CNN does not work very well since it requires enough training examples to learn its weights reliably and therefore for the CNN we used the full dataset. The results show that CNN has much higher accuracy than the other methods. The other methods are just a little better than random chance in terms of accuracy. This clearly shows that when the problem is more complex, we need to learn complex parameterizations that cannot be specified with simple MLN weights. Further, it also shows that modeling the correlations is particularly important for complex problems since EC yields poor accuracy for this task.

For the Images dataset, the structure of the MLN was equivalent to a logistic regression model since it connects features directly to the query. Therefore, when using Tuffy or Magician, the results obtained are similar to ones obtained by EC and therefore, we do not show this in the table. This problem is easier than the other problems in our evaluation since the correlations are easier to find in image data. Specifically, neighboring pixels will be close to each other in the embedding. However, the high accuracy of our approach here illustrates an important result that our approach captures correlations in sync with the correlations in natural images. Further, it also illustrates that without capturing these correlations as shown by the results for EC, generalization of the model is considerably reduced.

For the Movielens dataset, the results are very similar and the accuracy of CNN is again much higher than the other models. As with other datasets, the performance of purely MLN-based approaches are significantly worse than our proposed approach.

Scalability. Due to the CNN learning, our approach when learned with very few atoms is unable to accurately learn the weights. Thus, it is more suited to complex tasks, where the structure is hard to specify and that have a significant amount of training data. Fig. 2 (a) shows the learning time for the Reviews dataset as we vary the number of reviews. As we can see, even for 200K reviews the training time is quite small (around half an hour).

In general for small datasets (small number of atoms), our approach has poor accuracy. This is because learning the CNN parameters requires a reasonably large number of instances. As we see from Fig. 2 (b), as the number of reviews increase, which causes a corresponding increase in the number of atoms, the accuracy of our model improves. Of-course the trade-off is that the training time increases as well. However, we were able to perform fairly accurate predictions (ROC-AUC scores > 80) over datasizes of over 100K reviews. This is somewhat different from traditional MLN-based learning, where even a small increase to the number of atoms may result in learning becoming infeasible. Thus our model can handle orders of magnitude larger datasets than MLN-learning systems. The underlying difference is that, in MLN-learning we treat all the atoms as a single instance. Here, we treat atoms as independent instances, but at the same time the embedding neighborhood captures the most important relationships between other atoms. This shows that even in a very large space of atoms, by taking advantage of correlations, we do not need to consider the relationships between all atoms (which is intuitively what an MLN weight learning algorithm does) to learn expressive and accurate models. This helps us scale up to large datasets without sacrificing accuracy of predictions.

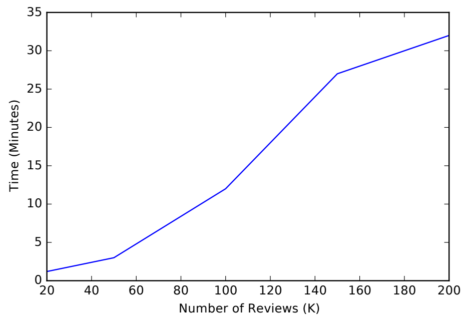
5.3 Comparison with Other Relational Learners

We compared our approach with other relational learning methods for the task of knowledge-base completion. We use a subset of entities and relations, $FB15k$ containing 592,213 triplets with 14,951 entities and 1,345 relationships from [3]. Triplet (h, r, t) can be simplified to $(h, r) \wedge (r, t) \wedge (h, t)$ that we use to create a simple MLN having formula $\texttt{Head\_Rel}(h, r) \wedge \texttt{Tail\_Rel}(t, r) \wedge \texttt{Head\_Tail}(h, t)$. We use this MLN to generate embeddings for entities and relations. Depending on the cardinalities between heads and tails, [3] categorizes the relationships into four classes: 1-TO-1, 1-TO-Many, Many-TO-1 and Many-TO-Many. Hence, we divide the triplets into equal sized four categories and for each category, we train a separate model and predict marginals of ground atoms. For each test triplet (h_i, r, t_j), we replace the head h_i with entities $h_1, h_2, h_2...$ from the domain of entities Δ_e for predicting heads while we replace the tails for predicting tails. We predict the marginal if replaced triplet $(h, r, t_j) \notin$ training triples. We multiply the marginals of $\texttt{Head\_Rel}(h, r)$, $\texttt{Tail\_Rel}(t_j, r)$ and $\texttt{Head\_Tail}(h, t_j)$, sort the values by decreasing order to compute the ranking.

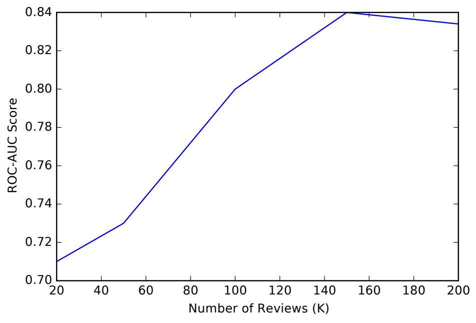
Table 2 shows the comparison with several baselines, `TransE` [3] that models relationships as translation of entities into embedding space, `Unstructured` [31] which is another mono-relational version of `TransE`, `RESCAL` that uses collective matrix factorization approach and energy based models `SE` [4], `SME`(linear) and `SME`(bilinear) [3]. We used the similar ranking procedure followed in [4]. Although our `CNN` based approach shows somewhat similar performance for 1-TO-1 relational triples, however, it significantly outperforms all other baselines for 1-TO-Many, Many-TO-1 and Many-TO-Many relationships. We also computed the ROC-AUC scores for our models (these scores are not published for other models) and we obtained an overall ROC-AUC score of 0.93 while EC had a score of 0.82.

6 Conclusion

We developed a model based on CNNs to perform reasoning in relational domains. To incorporate relational



(a)



(b)

Figure 2: (a) Learning time for the Reviews dataset for varying number of reviews. (b) Accuracy for the Reviews dataset for varying number of reviews.

	Predicting heads				Predicting tails			
Task	**1-TO-1**	**1-TO-MANY**	**Many-TO-1**	**Many-TO-Many**	**1-TO-1**	**1-TO-MANY**	**Many-TO-1**	**Many-TO-Many**
Unstructured	34.5	2.5	6.1	6.6	34.3	4.2	1.9	6.6
SE	35.6	62.6	17.1	37.5	34.9	14.6	68.3	61.3
SME(Linear)	35.1	53.7	19	40.3	32.7	14.9	61.6	43.3
SME(Bilinear)	30.9	69.6	19.9	38.6	28.2	13.1	76	41.8
TransE	**43.7**	65.7	18.2	47.2	**43.7**	19.7	66.7	50
CNN	40.3	**73.1**	**23.8**	**65.5**	39	**25.4**	**71.6**	**64.8**

Table 2: Comparison of Hits@10 (in %) on FB15k.

dependencies into the model, we learned the CNN on variables that have symmetrical relationships. However, while relational learning methods such as those based on MLNs learn jointly from the full relational database, in our model, we need to split the database into several independent instances and therefore loose some dependencies. To handle this, instead of a single set of parameters, we learned a distribution over the kernel functions using dropout. Our results on several problems showed that our proposed approach compares favorably in terms of accuracy with sate-of-the-art relational learning methods.

Future work includes developing generative models for relational data, explanations or interpretations of results of predictions made by our model, etc.

References

[1] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *J. Mach. Learn. Res.*, 18(1):3846–3912, January 2017.

[2] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 3:463–482, 2003.

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.

[4] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *AAAI*, pages 301–306, 2011.

[5] Jeffrey Chan, Valerio Perrone, Jeffrey Spence, Paul Jenkins, Sara Mathieson, and Yun Song. A likelihood-free inference framework for population genetic data using exchangeable neural networks. In *Neural Information Processing Systems*, pages 8594–8605. 2018.

[6] William W. Cohen. Tensorlog: A differentiable deductive database. *CoRR*, abs/1605.06523, 2016.

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

[8] Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with Bernoulli approximate variational inference. In *ICLR workshop track*, 2016.

[9] Robert Gens and Pedro M. Domingos. Deep symmetry networks. In *Neural Information Processing Systems*, pages 2537–2545, 2014.

[10] Mohammad Maminur Islam, Somdeb Sarkhel, and Deepak Venugopal. On lifted inference using neural embeddings. In *AAAI*, pages 7916–7923, 2019.

[11] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 219–230, 2008.

[12] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Neural Information Processing Systems*, pages 4289–4300, 2018.

[13] Tushar Khot, Niranjan Balasubramanian, Eric Gribkoff, Ashish Sabharwal, Peter Clark, and Oren Etzioni. Exploring markov logic networks for question answering. In *EMNLP*, pages 685–694, 2015.

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

[15] Xingyi Li, Fuxin Li, Xiaoli Z. Fern, and Raviv Raich. Filter shaping for convolutional neural networks. In *ICLR*, 2017.

[16] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *CoRR*, abs/1805.10872, 2018.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[18] Happy Mittal, Ayush Bhardwaj, Vibhav Gogate, and Parag Singla. Domain-size aware markov logic networks. In *AISTATS*, pages 3216–3224, 2019.

[19] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing YAGO: scalable machine learning for linked data. In *WWW*, pages 271–280, 2012.

[20] Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *PVLDB*, 4(6):373–384, 2011.

[21] D. Poole. First-Order Probabilistic Inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 985–991, Acapulco, Mexico, 2003. Morgan Kaufmann.

[22] Shebuti Rayana and Leman Akoglu. Yelp Dataset for Anomalous Reviews. Technical report, Stony Brook University, 2015. http://odds.cs.stonybrook.edu.

[23] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, 2010.

[24] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NIPS*, pages 3791–3803, 2017.

[25] Luciano Serafini and Artur S. d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. In *NeSy@HLAI*, volume 1768 of *CEUR Workshop Proceedings*, 2016.

[26] Jude W. Shavlik and Geoffery G. Towell. An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1(3):231–253, 1989.

[27] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intell.*, 46(1-2):159–216, November 1990.

[28] G. G. Towell and J. W. Shavlik. Knowledge-Based Artificial Neural Networks. *Artificial Intelligence*, 70:119–165, 1994.

[29] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, pages 2071–2080, 2016.

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

[31] Antoine Bordes Xavier Glorot and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. In *Machine Learning 94*, page 233259, 2014.

[32] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, pages 5498–5507, 2018.

[33] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2014.