RelNN: A Deep Neural Model for Relational Learning

Seyed Mehran Kazemi and **David Poole**
University of British Columbia
Vancouver, Canada
{smkazemi, poole}@cs.ubc.ca

Abstract

Statistical relational AI (StarAI) aims at reasoning and learning in noisy domains described in terms of objects and relationships by combining probability with first-order logic. With huge advances in deep learning in the current years, combining deep networks with first-order logic has been the focus of several recent studies. Many of the existing attempts, however, only focus on relations and ignore object properties. The attempts that do consider object properties are limited in terms of modelling power or scalability. In this paper, we develop relational neural networks (RelNNs) by adding hidden layers to relational logistic regression (the relational counterpart of logistic regression). We learn latent properties for objects both directly and through general rules. Back-propagation is used for training these models. A modular, layer-wise architecture facilitates utilizing the techniques developed within deep learning community to our architecture. Initial experiments on eight tasks over three real-world datasets show that RelNNs are promising models for relational learning.[1]

Introduction

Multi-relational data cannot be directly fed into traditional machine learning approaches such as logistic regression, SVMs, random forests, etc. Statistical relational AI (StarAI) (De Raedt et al. 2016) aims at developing models that work directly with relational data by capturing the interdependence among properties of the objects and the relationships they are involved in.

In relational data, there are usually several classes of objects, each class has certain properties defined for its members, and there are (various) relationships among the objects. Many works on learning from relational data focus only on predicting new relationships among objects given known relationships, and ignore predicting object properties. For the works that do consider predicting object properties, a recent comparative study shows that none of them perform well and that this problem is still only poorly understood (Kazemi et al. 2017). In this paper, we focus (primarily) on predicting a property of the objects in one class based on the rest of the data (e.g., predicting the gender of people given the movies they like). This problem is challenging when the property of each object in the class depends on a varying number of other objects' properties and relationships. In the StarAI community, this problem is known as *aggregation*.

With numerous advances in learning deep networks for many different applications, using deep neural approaches for relational data has been the focus of several recent studies. In this paper, we develop a framework for learning first-order deep neural models to learn from and reason with relational data. Our model is developed through deepening relational logistic regression (RLR) models (Kazemi et al. 2014), the directed analogue of Markov logic (Richardson and Domingos 2006), by enabling them to learn latent object properties both directly and through general rules, and connecting multiple RLR layers as a graph to produce relational neural networks (RelNNs). Similar to (Niepert, Ahmed, and Kutzkov 2016; Pham et al. 2017), we identify the relationship between our model and convolutional neural networks (ConvNets). Identifying this relationships allows RelNNs to be understood and implemented using well-known ConvNet primitives. Each training iteration of RelNNs is order of the amount of data times the size of the RelNN, making RelNNs highly scalable.

We evaluate RelNNs on three real-world datasets and compare them to well-know relational learning algorithms. We show how RelNNs address a relational learning issue raised in (Poole et al. 2014) who showed that as the population size increases, the probabilities of many variables go to 0 or 1, making the model over-confident. Obtained results indicate that RelNNs are promising models for relational learning.

Relational Logistic Regression and Markov Logic Networks

StarAI models aim at modelling the probabilities about relations among objects. Before knowing anything about the objects, these models treat them identically and apply tied parameters to them. In order to describe these models, first we need to introduce some definitions and terminologies.

A **population** is a finite set of **objects**. **Logical variables** (logvars) start with lower-case letters, and **constants** denoting objects start with upper-case letters. Associated with a logvar x is a population Δ_x with $|\Delta_x|$ representing the cardi-

[1]Code: https://github.com/Mehran-k/RelNN

nality of the population. A lower-case letter in bold refers to a tuple of logvars. An upper-case letter in bold refers to a tuple of constants. An **atom** is of the form $\mathsf{S}(t_1, \ldots, t_k)$ where S is a predicate symbol and each t_i is a logvar or a constant. We write a **substitution** as $\theta = \{\langle x_1, \ldots, x_k\rangle / \langle t_1, \ldots, t_k\rangle\}$ where each x_i is a different logvar and each t_i is a logvar or a constant in Δ_{x_i}. A **grounding** of an atom with logvars $\{x_1, \ldots, x_k\}$ is a substitution $\theta = \{\langle x_1, \ldots, x_k\rangle / \langle X_1, \ldots, X_k\rangle\}$ mapping each of its logvars x_i to an object $X_i \in \Delta_{x_i}$. Given a set $\mathcal{A}$ of atoms, we denote by $\mathcal{G}(\mathcal{A})$ the set of all possible groundings for the atoms in $\mathcal{A}$. A **literal** is an atom or its negation. A **formula** φ is a literal, a disjunction $\varphi_1 \vee \varphi_2$ of formulas or a conjunction $\varphi_1 \wedge \varphi_2$ of formulas. A **conjunctive formula** has no disjunctions. Applying a **substitution** $\theta = \{\langle x_1, \ldots, x_k\rangle / \langle t_1, \ldots, t_k\rangle\}$ on a formula φ (written as $\varphi\theta$) replaces each x_i in φ with t_i. A **weighted formula (WF)** is a tuple $\langle \varphi, w\rangle$ where φ is a formula and w is a weight.

Let $\mathsf{Q}(\mathbf{x})$ be an atom whose probability depends on a set $\mathcal{A}$ (not containing Q) of atoms (called the parents of Q), ψ be a set of WFs containing only atoms from $\mathcal{A}$, $\hat{I}$ be a function from groundings in $\mathcal{G}(\mathcal{A})$ to truth values, $\mathbf{X}$ be an assignment of objects to $\mathbf{x}$, and $\theta = \{\mathbf{x}/\mathbf{X}\}$. **Relational logistic regression (RLR)** (Kazemi et al. 2014) defines the probability of $\mathsf{Q}(\mathbf{X})$ given $\hat{I}$ as follows:

$$Prob(\mathsf{Q}(\mathbf{X}) = True \mid \hat{I}) = sigmoid(sum) \tag{1}$$

where,

$$sum = \sum_{\langle \varphi, w\rangle \in \psi} w * \eta(\varphi\theta, \hat{I}) \tag{2}$$

where $\eta(\varphi\theta, \hat{I})$ is the number of instances of $\varphi\theta$ that are true w.r.t. $\hat{I}$. Note that $\eta(\mathsf{True}, \hat{I}) = 1$. Following (Kazemi et al. 2014), w.l.o.g we assume the formulae of all WFs for RLR models are conjunctive.

Markov logic networks (MLNs) (Richardson and Domingos 2006) use WFs to define a probability distribution over ground atoms. As shown in (Poole et al. 2014), when all groundings in $\mathcal{G}(\mathcal{A})$ are observed, an RLR model is identical to an MLN with corresponding WFs. We use RLR/MLN when the two models are identical.

Example 1. Let Happy(x) be an atom which depends on $\mathcal{A} = \{\mathsf{Friend}(x, y), \mathsf{Kind}(y)\}$, and let $\hat{I}$ be a function from $\mathcal{G}(\mathcal{A})$ to truth values. Let an RLR/MLN model define the conditional probability of Happy using WFs in Fig 1. According to this model: $Prob(\mathsf{Happy}(X) = True \mid \hat{I}) = sigmoid(-4.5 + 1 * \eta(\mathsf{Friend}(y, X) \wedge \mathsf{Kind}(y), \hat{I}))$, where $\eta(\mathsf{Friend}(y, X) \wedge \mathsf{Kind}(y), \hat{I}) = \#Y \in \Delta_y$ s.t. $\mathsf{Friend}(Y, X) \wedge \mathsf{Kind}(Y)$ according to $\hat{I}$, corresponding to the number of friends of X that are kind. When this count is greater than or equal to 5, the probability of X being happy is closer to one than zero; otherwise, the probability is closer to zero than one. Therefore, the two WFs model "someone is happy if they have at least 5 friends that are kind".

Continuous atoms: The RLR model defined above only works with Boolean or multi-valued parents. In order to

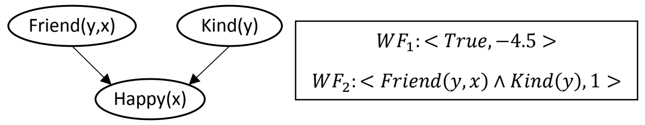


Figure 1: An RLR model taken from (Kazemi et al. 2014).

allow for parents with continuous atoms, we use Fatemi, Kazemi, and Poole (2016)'s proposal. If True and False are associated with 1 and 0, $\wedge$ in WFs can be substituted with $*$. Then continuous atoms may be used in WFs. For example if for some $X \in \Delta_x$ we have $\hat{I}(\mathsf{R}(X)) = 1$, $\hat{I}(\mathsf{S}(X)) = 0$, and $\hat{I}(\mathsf{T}(X)) = 0.2$, then $\langle w, \mathsf{R}(X) * \mathsf{S}(X)\rangle$ evaluates to 0, $\langle w, \mathsf{R}(X) * \neg\mathsf{S}(X)\rangle$ evaluates to w, and $\langle w, \mathsf{R}(X) * \neg\mathsf{S}(X) * \mathsf{T}(X)\rangle$ evaluates to $0.2 * w$.

Relational Neural Networks

We encode RLR/MLN models with a layer-wise architecture by designing relational counterparts of the linear layer (LL), activation layer (AL), and error layer (EL) in neural networks. This enables building relational neural nets (RelNNs) by connecting several relational LLs and relational ALs as a graph. It also enables utilizing the *back propagation* algorithm for training these models by adding a relational EL at the end of the graph, feeding the data forward, calculating the prediction errors, back propagating the derivatives with respect to errors, and updating the parameters.

Let $\mathcal{A}_{in}$ and $\mathcal{A}_{out}$ represent two distinct sets of atoms, and E represent an error function. Let $\hat{I}$ be a function from $\mathsf{P}(\mathbf{X}) \in \mathcal{G}(\mathcal{A}_{in})$ to values. Let $\hat{O}$ be a function from $\mathsf{Q}(\mathbf{X}) \in \mathcal{G}(\mathcal{A}_{out})$ to values. Let $D_{\hat{O}}$ be a function from $\mathsf{Q}(\mathbf{X}) \in \mathcal{G}(\mathcal{A}_{out})$ to $\frac{\partial E}{\partial \mathsf{Q}(\mathbf{X})}$. Let $D_{\hat{I}}$ be a function from $\mathsf{P}(\mathbf{X}) \in \mathcal{G}(\mathcal{A}_{in})$ to $\frac{\partial E}{\partial \mathsf{P}(\mathbf{X})}$.

A *layer* is a generic box that given $\hat{I}$ as input, outputs $\hat{O}$ based on $\hat{I}$ and its internal structure, and given $D_{\hat{O}}$, updates its internal weights and outputs $D_{\hat{I}}$ using the chain rule.

A *relational linear unit* for an atom $\mathsf{Q}(\mathbf{x})$ is identical to the linear part of RLR in Eq. (2). A *relational LL (RLL)* consists of $t > 0$ *relational linear units* with the j-th unit having atom $\mathsf{Q_j}(\mathbf{x_j})$ and a set $\psi_j = \{\langle w_{j1}, \varphi_{j1}\rangle, \langle w_{j2}, \varphi_{j2}\rangle, \ldots, \langle w_{jm_j}, \varphi_{jm_j}\rangle\}$ of m_j WFs. $\mathcal{A}_{in}$ for an RLL contains all atoms in ψ_js and $\mathcal{A}_{out}$ contains all $\mathsf{Q_j}(\mathbf{x_j})$s. $\hat{I}$ comes from input data or from the layers directly connected to the RLL. For any $\mathsf{Q}(\mathbf{X}) \in \mathcal{G}(\mathcal{A}_{out})$, $\hat{O}(\mathsf{Q}(\mathbf{X}))$ is calculated using the linear part of Eq. (1) with respect to $\hat{I}$ and ψ_j. An RLL can be seen as general rules with tied parameters applied to every object.

Example 2. Consider an RLL with $\mathcal{A}_{in} = \{\mathsf{Friend}(x, y), \mathsf{Kind}(y)\}$, $\mathcal{A}_{out} = \{\mathsf{Happy}(x)\}$, and with the WFs in Example 1. Suppose there are 10 objects: $\Delta_x = \{X_1, X_2, \ldots, X_{10}\}$. Let $\hat{I}$ be a function from $\mathcal{G}(\mathcal{A}_{in})$ to values according to which $X_1, X_2, \ldots, X_{10}$ have 3, 0, ..., and 7 friends that are kind respectively. $\hat{O}$ for this RLL is a function from groundings in $\mathcal{G}(\mathcal{A}_{out})$

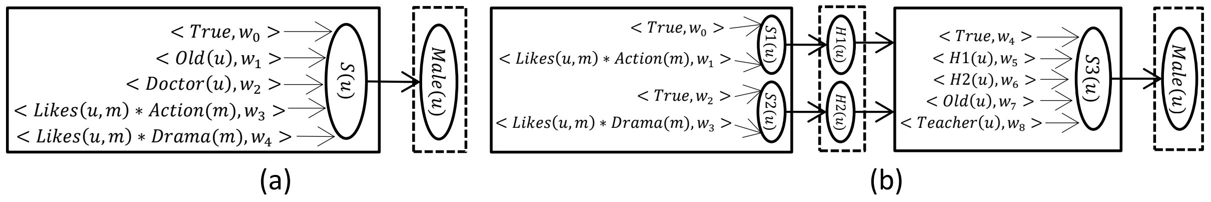


Figure 2: An RLR and a RelNN models for predicting the gender of users in a movie rating system with a layer-wise architecture.

to values as $\hat{O}(\mathsf{Happy}(X_1)) \rightarrow -1.5, \hat{O}(\mathsf{Happy}(X_2)) \rightarrow -4.5, \ldots, \hat{O}(\mathsf{Happy}(X_{10})) \rightarrow 2.5$.

Consider the j-th relational linear unit. For each assignment of objects $\mathbf{X_{ju}}$ to $\mathbf{x_j}$, let $\theta_{ju} = \{\mathbf{x_j}/\mathbf{X_{ju}}\}$. The derivative with respect to each weight w_{jk} can be calculated as $\sum_{\mathbf{X_{ju}}} \eta(\varphi_{jk}\theta_{ju}, \hat{I}) * D_{\hat{O}}(Q_j(\mathbf{X_{ju}}))$. To show how $D_{\hat{I}}$ is calculated, for ease of exposition we assume no predicate appears twice in a formula (i.e. no self-joins). It is straight-forward to relax this assumption. Let $\varphi \backslash \mathsf{P}$ represent formula φ with any atom with predicate P (or its negation) removed from it. For each assignment of objects $\mathbf{X_{iv}}$ to the logvars $\mathbf{x_i}$ of P, let $\theta_{iv} = \{\mathbf{x_i}/\mathbf{X_{iv}}\}$. Then $D_{\hat{I}}(\mathsf{P}(\mathbf{X_{iv}})) = \sum_j \sum_{\mathbf{X_{ju}}} \sum_{k=1,\mathsf{P}\in\varphi_{jk}}^{m_j} w_{jk} * \eta(((\varphi_{jk} \backslash \mathsf{P})\theta_{ju})\theta_{iv}, \hat{I}) * D_{\hat{O}}(\mathsf{Q_j}(\mathbf{X_{ju}}))$.

The *relational AL (RAL)* and *relational EL (REL)* are similar to their non-relational counterparts. RAL applies an activation function A (e.g., $sigmoid$, $tanh$, or $ReLU$) to its inputs and outputs the activations. REL compares the inputs to target labels, finds the prediction errors based on an error function E, and outputs the prediction errors made by the model for each target object.

A **relational neural network (RelNN)** is a structure containing several RLL and RALs connected to each other as a graph. In our experiments, we consider the activation function to be the sigmoid function. During training, an REL is also added at the end of the sequence. Fig 2 represents an example of an RLR and a simple RelNN model for predicting the gender of users based on their age, occupation, and the movies they like. We use boxes with solid lines for RLLs and boxes with dashed lines for RALs. For the first RLL, $\mathcal{A}_{in} = \{\mathsf{Likes}(u,m), \mathsf{Action}(m), \mathsf{Drama}(m)\}$ and all values in $\hat{I}$ come from the observations. For this RLL, $\mathcal{A}_{out} = \{\mathsf{S1}(u), \mathsf{S2}(u)\}$. For the second RLL, $\mathcal{A}_{in} = \{\mathsf{Old}(u), \mathsf{Teacher}(u), \mathsf{H1}(u), \mathsf{H2}(u)\}$ and $\hat{I}$ for the first two atoms comes from observations, and for the last two comes from the layers directly connected to the RLL. Fig 3 shows a more complicated RelNN structure for gender predicting in PAKDD-15 competition.

Motivations for hidden layers

Motivation 1. Consider the model in Fig 2(a) and let n_U represent the number of action movies that user U likes. As Poole et al. (2014) point out, if the probability of maleness depends on n_U, as n_U increases the probability of maleness either goes to 0 or 1. This causes the model to become over-confident of its predictions when there exists many ratings and do a poor job, especially in terms of log loss. This is, however, not the case for RelNNs. In the RelNN model in Fig 2(b), the probability of maleness depends on $w_5 * sigmoid(w_0 + w_1 n_U)$. w_1 controls the steepness of the slope of the sigmoid, w_0 moves the slope around, and w_5 controls the importance of the slope in the final prediction. With this model, as n_U increases, the probability of maleness may converge to any number in the $[0, 1]$ interval. Furthermore, this model enables learning of the critical zones.

Motivation 2. Suppose the underlying truth for the models in Fig 2 is that males correspond to users who like at least p action movies such that each movie has less than q likes. A typical RLR/MLN model cannot learn such a model because first it needs to learn something about movies (i.e. having less than q likes), combine it with being *action* and count them. However, this can be done using a RelNN (the hidden layer should contain a relational linear unit with atom $\mathsf{H}(m)$ and WFs $\langle w_1, \mathsf{Likes}(u,m)\rangle$ and $\langle w_2, \mathsf{True}\rangle$). Thus, hidden layers increase the modelling power by enabling the model to learn generic rules and categorize the objects accordingly, then treat objects in different categories differently.

Motivation 3. Kazemi et al. (2014) show how different types of existing explicit aggregators can be represented using RLR. However, some of those cases (e.g., noisy-or and mode $= t$) require two RLLs and two RALs, i.e. RelNNs.

Learning latent properties directly

Objects may contain latent properties that cannot be specified using general rules, but can be learned directly from data during training. Such properties have proved effective in many tasks (e.g., recommendation systems (Koren, Bell, and Volinsky 2009)). These properties can be also viewed as soft clustering the objects into different categories. We call the latent properties specifically learned for each object *numeric latent properties* and the general latent properties learned through WFs in RLLs *rule-based latent properties* to distinguish between the two. In the RelNN in Fig 3, for instance, $\mathsf{S1}(i)$, $\mathsf{S4}(u)$ and $\mathsf{S7}(u)$ are the outputs of RALs and so are rule-based latent properties whereas $\mathsf{N2}(b)$ and $\mathsf{N4}(a)$ are numeric latent properties.

Consider the models in Fig 2 and let $\mathsf{Latent}(m)$ be a numeric latent property of the movies whose value is to be learned during training. One may initialize $\mathsf{Latent}(m)$ with random values for each movie and add a WF $\langle w, \mathsf{Likes}(u,m) * \mathsf{Latent}(m)\rangle$ to the first RLL. As mentioned before, during the back propagation phase, an RLL provides the derivatives with respect to each of the inputs.

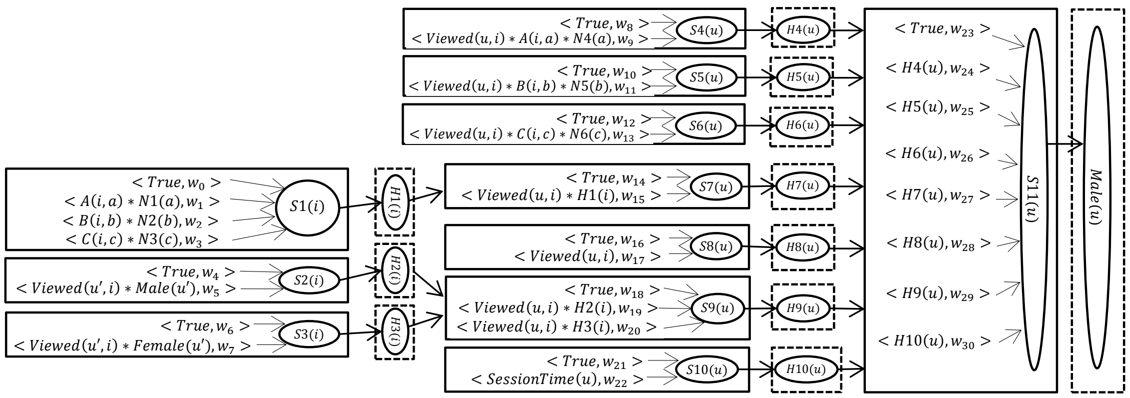


Figure 3: RelNN structure for predicting the gender in PAKDD15 dataset.

This means for each grounding $\mathsf{Latent}(M)$, we will have $D_{\hat{I}}(\mathsf{Latent}(M))$. Therefore, these numeric latent values can also be updated during learning using gradient descent.

From ConvNet Primitives to RelNNs

Niepert, Ahmed, and Kutzkov (2016) and Pham et al. (2017) explain why their relational learning models for graphs can be viewed as instances of ConvNet. We explain why RelNNs can also be viewed as an instance of ConvNets. Compared to prior work, we go into more details and provide more intuition. Such a connection offers the benefit of understanding and implementing RelNNs using ConvNet primitives.

The cells in input matrices of ConvNets (e.g., image pixels) have spatial correlation and spatial redundancy: cells closer to each other are more dependent than cells farther away. For instance if M represents an input channel of an image, the dependence between $M[i, j]$ and $M[i + 1, j + 1]$ may be much more than the dependence between $M[i, j]$ and $M[i, j + 20]$. To capture this type of dependency, convolution filters are usually small squared matrices. Convolution filters contain tied parameters so different regions of the input matrices are treated identically.

For relational data, the dependencies in the input matrices (the relationships) are different: the cells in the same row or column (i.e. relationships of the same object) have higher dependence than the cells in different rows and columns (i.e. relationships of different objects). For instance for a matrix L representing which users like which movies, the dependence between $L[i, j]$ and $L[i+1, j+1]$ (different people and movies) may be far less than the dependence between $L[i, j]$ and $L[i, j+20]$ (same person and different movies). A priori, all rows and columns of the input matrices are exchangeable. Therefore, to adapt ConvNets for relational data, we need vector-shaped filters that are invariant to row and column swapping and better capture the relational dependence and the exchangeability assumption.

One way to modify ConvNets to be applicable to relational data is as follows. We consider relationships as inputs to the network. We apply vector-shaped convolution filters on the rows and columns of the relationship matrices. For instance for gender prediction from movie ratings, a convolution filter may be a vector with size equal to the number of the movies. The values of these filters can be learned during training (like ConvNet filters), or can be fixed in which case they correspond to observed properties (e.g., a vector representing which movies are action movies). Convolving each filter with each matrix produces a vector. These vectors go through an activation layer and are used as a filter in the next layers. Besides convolving a filter with a matrix, one can join two matrices and produce a new matrix for the next layer. Joining two matrices $R(x, m)$ and $T(m, a)$ produces a new matrix $S(x, a)$ where for some $X \in x$ and $A \in a$, $S(X, A) = \sum_{M \in m} R(X, M) * T(M, A)$. The filters in the next layers can be applied to either input matrices, or the ones generated in previous layers. By constructing an architecture and training the network, the learned filters identify low and high level features from the input matrices.

While these operations are modifications of ConvNet operations, they all correspond to the operations in our RLR/MLN perspective. Vector-shaped filters correspond to including numeric latent properties in WFs. Fixed filters correspond to including observed atoms with a single logvar in WFs. Joining matrices corresponds to using two binary atoms in a WF.

Example 3. Suppose we want to predict the gender of users in a movie rating system based on the movies they like, whether the movie is action or not, and whether the user is old or not. Fig 4 shows one layer of a RelNN for this problem in terms of ConvNet operations. The binary relation $\mathsf{Likes}(u, m)$ is considered as the input. Three filters have been applied to this matrix: one filter with fixed values corresponding to $\mathsf{Action}(m)$, one filter with fixed values corresponding to $\mathsf{Old(u)}$, and one filter to be learned corresponding to a numeric latent property $\mathsf{N}(m)$. Note that $\mathsf{Action}(m)$ and $\mathsf{N}(m)$ are row-shaped filters and $\mathsf{Old}(u)$ is column-shaped. Convolving these filters with $\mathsf{Likes}(u, m)$ and sending the resulting vectors through an activation layer is equivalent to having an RLL with three relational linear units: the first unit contains:

$$\langle \mathsf{True}, w_0 \rangle$$
$$\langle \mathsf{Likes}(u, m) * \mathsf{Action}(m), w_1 \rangle$$

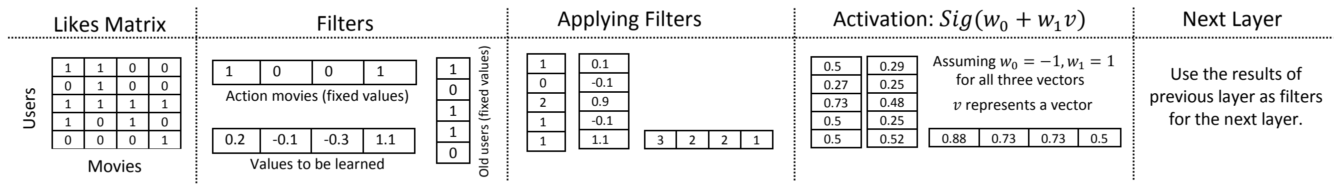

Likes Matrix
Filters
Applying Filters
Activation: $Sig(w_0 + w_1 v)$
Next Layer
Users
Movies
Action movies (fixed values)
Values to be learned
Old users (fixed values)
Assuming $w_0 = -1, w_1 = 1$ for all three vectors
v represents a vector
Use the results of previous layer as filters for the next layer.

Figure 4: An example of a layer of a RelNN demonstrated with ConvNet operations.

and the other ones have $\mathsf{Old}(\mathsf{u})$ and $\mathsf{N}(m)$ instead of $\mathsf{Action}(m)$ respectively. The obtained vectors after activation can be used as filters for the next layers, or for making the final predictions.

Empirical Results

In our experiments, we tend to answer these questions: **Q1:** how does RelNN's performance compare to other well-known relational learning algorithms, **Q2:** how numeric and rule-based latent properties affect the performance of the RelNNs, and **Q3:** how well RelNNs extrapolate to unseen cases and address the population size issue pointed out in (Poole et al. 2014) and discussed in Motivation 1.

Datasets: We use three real-world datasets in our experiments. Our first dataset is the Movielens 1M dataset (Harper and Konstan 2015) ignoring the actual ratings and only considering if a movie has been rated or not, and considering only action and drama genres. Our second dataset is from PAKDD15 gender prediction competition[2] but we only considered the A, B, and C prefixes of the items and ignored the D prefix because each D prefix is on average seen by ≈ 1.5 people. We also ignored the information within the sequence of the items viewed by each user. Our third dataset contains all Chinese and Mexican restaurants in Yelp dataset challenge[3] (ignoring the ones that have both Chinese and Mexican foods), and the task is to predict if a restaurant is Chinese or Mexican given the people who reviewed them, and whether they have fast food and/or seafoods or not.

Learning algorithms and learning methodology: Traditional machine learning approaches are not directly applicable to our problems. Also due to the large size of the datasets, some relational models do not scale to our problems. For instance, we tried Problog (De Raedt, Kimmig, and Toivonen 2007), but the software crashed after running for a few days even for simple models. The reason for scalability issues with many existing models (e.g., MLNs and Problog) is because they consider probabilistic (instead of neural) units and do inference and learning using EM. It took a day for an analogous EM-based implementation of RelNNs to learn a model for a synthetic dataset with 100 relations, whereas our current model (which is neural and uses back propagation) took under an hour for a million relations. We tried several baselines and only reported the ones that performed best on our datasets. Kazemi et al. (2017) explain why each of the existing baselines for aggregation perform poorly and why this problem is still poorly understood.

In our baselines, *mean* corresponds to always predicting the mean of the training data. The *matrix factorization* corresponds to the best performing matrix factorization model according to Kazemi et al. (2017)'s experiments: the relation matrix is factorized into object latent properties using the factorization technique in (Koren, Bell, and Volinsky 2009), then, using Weka (Hall et al. 2009), a linear/logistic regression model is learned over the latent and observed properties of the target objects. We also tried several other variants (including RESCAL's algorithm (Nickel, Tresp, and Kriegel 2012)), but Kazemi et al. (2017)'s model was the best performing. The *RDN-Boost* (Natarajan et al. 2012) model can be viewed as an extension of random forests with ways to aggregate over multiple observations, thus enabling random forests to be applicable to relational data. The *k-nearest neighbors collaborative filtering* model finds k objects with observed labels that are similar to the target object in terms of the relationships they participate in, creates a feature as the weighted mean of the labels of the similar objects, and feeds this label along with other features of the target object to a classification/regression model. For MovieLens and Yelp datasets, collaborative filtering produces one feature and for PAKDD dataset, it produces three features, one for each item prefix. We used cosine similarity function. The value of k was set using cross validation. Similar to matrix factorization, linear and logistic regression of Weka were used for classification and regression.

For all RelNN and RLR/MLN models in our experiments, we used fixed structures (i.e. fixed set of WFs and connections among them) and learned the parameters using back-propagation with multiple restarts. The structure of the RelNN model for Movielens dataset is that of Fig 2(b) extended to include all ages and occupations with 2 numeric latent properties, 3 RLLs and 3 RALs. The structure of the RelNN model used for PAKDD15 and Yelp datasets is as in Fig 3 and Fig 5 respectively. We leave the problem of learning these structures automatically from data as future work. The structure of the RLR/MLN models are similar to the RelNN models but without hidden layers and numeric latent properties. Back propagation for the RLR/MLN case corresponds to the discriminative parameter learning of (Huynh and Mooney 2008). To avoid numerical inconsistencies when applying our models to PAKDD15 dataset, we assumed there exist a male and a female in the training set who have viewed all items. For all experiments, we split the data into 80/20 percent train/test.

[2]`https://knowledgepit.fedcsis.org/contest/view.php?id=107`

[3]`https://www.yelp.com/dataset_challenge`

Table 1: Performance of different learning algorithms based on accuracy, log loss, and MSE $\pm$ standard deviation on three different tasks. *NA* means the method is not directly applicable to the prediction task/dataset. The best performing method is shown in bold. For models where standard deviation was zero, we did not report it in the table.

		Learning Algorithm					
Task	Measure	Mean	Matrix Factorization	Collaborative Filtering	RDN-Boost	RLR/MLN	RelNN
MovieLens Gender	Accuracy	0.7088	0.7550 $\pm$ 0.0073	0.7510	0.7234	0.7073 $\pm$ 0.0067	**0.7902 $\pm$ 0.0051**
	Log Loss	0.8706	0.7318 $\pm$ 0.0073	0.7360	0.8143	0.8441 $\pm$ 0.0255	**0.6548 $\pm$ 0.0026**
	MSE	0.2065	0.1649 $\pm$ 0.0026	0.1675	0.1904	0.1987 $\pm$ 0.0067	**0.1459 $\pm$ 0.0009**
PAKDD Gender	Accuracy	0.7778	NA	0.8806	0.7778	0.8145 $\pm$ 0.0388	**0.8853 $\pm$ 0.0002**
	Log Loss	0.7641	NA	0.5135	0.8039	0.7224 $\pm$ 0.0548	**0.5093 $\pm$ 0.0037**
	MSE	0.1728	NA	0.1026	0.1842	0.1522 $\pm$ 0.0185	**0.1009 $\pm$ 0.0008**
Yelp Business Prediction	Accuracy	0.6168	0.6154 $\pm$ 0.0041	0.6712	0.6156	0.6168 $\pm$ 0.000	**0.6927 $\pm$ 0.0077**
	Log Loss	0.9604	0.9394 $\pm$ 0.0033	0.8757	0.9458	0.9435 $\pm$ 0.0034	**0.8531 $\pm$ 0.0090**
	MSE	0.2364	0.2300 $\pm$ 0.0012	0.2084	0.2316	0.2309 $\pm$ 0.0010	**0.2023 $\pm$ 0.0024**
MovieLens Age	MSE	156.0507	104.5967 $\pm$ 0.9978	90.8742	NA	156.0507 $\pm$ 0.000	**62.7000 $\pm$ 0.7812**

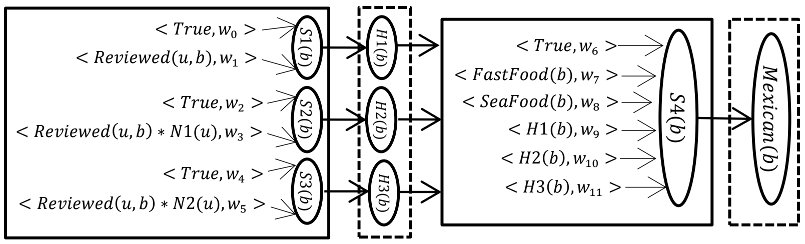


Figure 5: RelNN structure used for Yelp dataset. $\mathsf{N1}(u)$ and $\mathsf{N2}(u)$ are numeric latent properties.

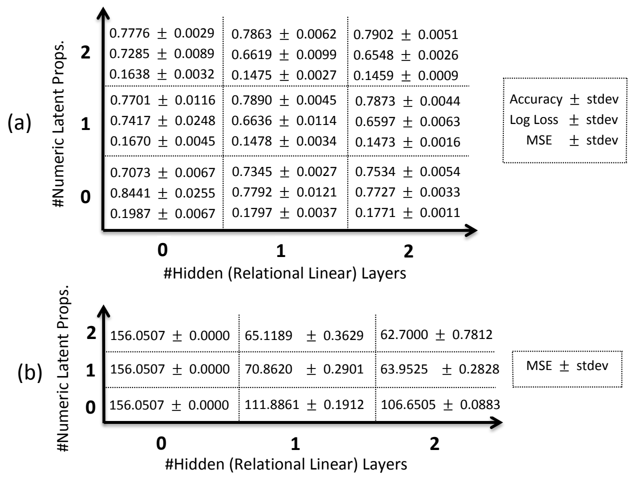
We imposed a Laplacian prior on all our parameters (weights and numeric latent properties). For classification, we further regularized our model predictions towards the mean of the training set using a hyper-parameter λ as: $Prob = \lambda * mean + (1 - \lambda) * (ModelSignal)$. This regularization alleviates the over-confidence of the model and avoids numerical inconsistencies arising when taking the logs of the predictions. Note that this regularization corresponds to adding an extra layer to the network. We reported the *accuracy* indicating the percentage of correctly classified instances, the *mean squared errors (MSE)*, and the *log loss*. We conducted each experiment 10 times and reported the mean and standard deviation.

Experiments: Our experiments include predicting the gender for the Movielens and PAKDD15 datasets, predicting the age for Movielens dataset, and predicting the type of food for the Yelp restaurants dataset. Even though the age is divided into 7 categories in the MovieLens dataset, we assume it is a continuous variable to see how RelNNs perform in predicting continuous variables. To make the categories more realistic, for people in age category/interval [i, j], we assume the age is $(i + j)/2$, i.e. the mean of the interval. For the first and last categories, we used 16 and 60 as the ages.

Table 1 compares RelNNs with other well-known relational learning algorithms as well as a baseline. It can be viewed from the table how RelNNs outperform well-known relational learning models in terms of all three performance metrics. Note that all baselines use the same observed features. Manual feature engineering from relations is very difficult, so the challenge for the models is to extract useful features from relations. As explained in (Kazemi et al. 2017), the problem with the baselines is with their lower modeling power and inappropriate implicit assumptions, and not with engineering features. The results in Table 1 answer **Q1**.

For **Q2**, we changed the number of hidden layers and numeric latent properties in RelNN to see how they affect the performance. The obtained results for predicting the gender and age in the Movielens dataset can be viewed in Fig 6. The results show that both hidden layers and numeric latent properties (especially the first one of each) have a great impact on the performance of the model. When hidden layers are added, as we conjectured in motivation 1, the log loss and MSE improve substantially as the over-confidence of the model decreases. Note that adding layers only adds a constant number of parameters, but adding k numeric latent properties adds $k * |\Delta_m|$ parameters. In Fig 6, a RelNN with 2 hidden layers and 1 numeric latent property has many fewer parameters than a MLN/RLR with no hidden layers and 2 numeric latent properties, but outperforms it.

For **Q3**, we conduct two experiments: 1- we train a RelNN on a large population, and test it on a small population, and 2- we train a RelNN on a small population and test it on a large population. The first experiment can be also seen as how severely each model suffers from the cold start prob-



(a)

#Numeric Latent Props. \ #Hidden (Relational Linear) Layers	0	1	2
2	0.7776 ± 0.0029 0.7285 ± 0.0089 0.1638 ± 0.0032	0.7863 ± 0.0062 0.6619 ± 0.0099 0.1475 ± 0.0027	0.7902 ± 0.0051 0.6548 ± 0.0026 0.1459 ± 0.0009
1	0.7701 ± 0.0116 0.7417 ± 0.0248 0.1670 ± 0.0045	0.7890 ± 0.0045 0.6636 ± 0.0114 0.1478 ± 0.0034	0.7873 ± 0.0044 0.6597 ± 0.0063 0.1473 ± 0.0016
0	0.7073 ± 0.0067 0.8441 ± 0.0255 0.1987 ± 0.0067	0.7345 ± 0.0027 0.7792 ± 0.0121 0.1797 ± 0.0037	0.7534 ± 0.0054 0.7727 ± 0.0033 0.1771 ± 0.0011

Accuracy ± stdev
Log Loss ± stdev
MSE ± stdev

(b)

#Numeric Latent Props. \ #Hidden (Relational Linear) Layers	0	1	2
2	156.0507 ± 0.0000	65.1189 ± 0.3629	62.7000 ± 0.7812
1	156.0507 ± 0.0000	70.8620 ± 0.2901	63.9525 ± 0.2828
0	156.0507 ± 0.0000	111.8861 ± 0.1912	106.6505 ± 0.0883

MSE ± stdev

Figure 6: Predicting (a) gender and (b) age on MovieLens using RelNNs with different number of numeric latent properties and hidden layers.

lem. The second experiment can be also seen as how well these models extrapolate to larger populations. For the first experiment, we trained two models for predicting the gender of the users in the MovieLens dataset both containing one numeric latent property, but one containing no hidden layers and the other containing one hidden layer. We only gave the first k ratings of the test users to the model and recorded the log loss of the two models. We did this for different values of k and plotted the results. Obtained results can be viewed in Fig 7. Note that since in MovieLens dataset all users have rated at least 20 movies, the model has not seen a case where a user has rated less than 20 movies. That is, the model has been trained on large population sizes (≥ 20). In this experiment, the cases in Fig 7 where $k < 20$ correspond to small populations (and cold start).

When $k = 0$ (i.e. we ignored all ratings of the test users), both models had almost the same performance ($logloss \approx -0.9176$)[4]. As soon as we add one rating for the test users, the performance of RelNN substantially improves, but the performance of the RLR/MLN model is not affected much. According to the plot, the gap between the two models is more when the test users have fewer ratings (i.e. unseen cases), but it gets less and becomes steady as the number of ratings increases and becomes closer to the number of ratings in the train set.

For the second experiment, for each user in the train set we kept only the first r ratings where $0 \leq r \leq 20$ was generated randomly for each user. For the new dataset, we repeated the previous experiment and obtained the diagram in Fig 8. It can be viewed in this diagram that the RLR/MLN model works best when we keep the first 15 ratings of the test users (i.e. $k = 15$), but for higher values of k, its performance starts to deteriorate. The performance of the RelNN

[4]This is not shown in the diagram in Fig 7 as the x-axis of the diagram is in log scale.

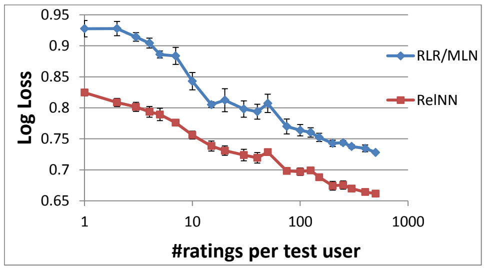


Figure 7: Results on predicting the gender when we only use the first k (on the x-axis) ratings of the test users and use all ratings in train set for learning.

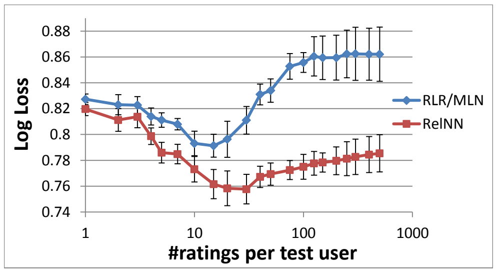


Figure 8: Results on predicting the gender when we only use the first k (on the x-axis) ratings of the test users and use $0 \leq r \leq 20$ ratings of each user (r is generated randomly for each user) in the train set for learning.

model, on the other hand, improves even until $k = 30$. After this point as k increases, the performance of both models deteriorates, but the gap between the two models becomes much more for larger values of k, as the RLR/MLN model becomes over-confident. These results validate the hypothesis in (Poole et al. 2014) that as the population size grows, RLR/MLN becomes over-confident and predicts with probabilities close to 0 and 1. The results also show how RelNNs address this issue and validate our Motivation 1.

Related Work

Recently, there has been a great body of research on learning from relational data using *tensor factorization*. These methods learn embeddings for each object and each relationship. The probability of two objects participating in a relation is a simple function of the objects' and relation's embeddings (e.g., the sum of the element-wise product of the embeddings). Well-known tensor factorization works include (Nickel, Tresp, and Kriegel 2012; Bordes et al. 2013; Socher et al. 2013; Neelakantan, Roth, and McCallum 2015; Trouillon et al. 2016) with a recent survey in (Nguyen 2017). Except RESCAL (Nickel, Tresp, and Kriegel 2012), all other works along this line only focus on predicting new relations from existing ones (cf. (Nickel et al. 2016) section X.A). As described in (Kazemi et al. 2017), for aggregation problems studied in this work, RESCAL's proposal ends up memorizing the training labels and does not generalize to unseen cases. The same issue exists for other tensor factorization algorithms thus making them unsuitable for aggregation.

Besides tensor factorization models, several other relational learning models only focus on predicting relations and ignore properties, or, relying on embeddings, suffer from the same issues as tensor factorization algorithms. Examples of these works include path-constrained random walks (e.g., (Lao and Cohen 2010; Lao, Mitchell, and Cohen 2011)), and several hybrid approaches (e.g., (Das et al. 2017; Rocktäschel and Riedel 2017; Yang, Yang, and Cohen 2017; Wang, Shi, and Yeung 2017)).

Several works on predicting object properties based on deep learning consider only a subset of relational data such as set data (Zaheer et al. 2017) or graph data (Pham et al. 2017) (where there can be multiple relationships among objects, but all objects belong to the same class). Several works on predicting object properties are based on recurrent neural networks (RNNs) (see e.g., (Uwents and Blockeel 2005; Moore and Neville 2017)). Zaheer et al. (2017) show that RNN based methods do not perform well for set data (a restricted form of relational data) as RNN-based models consider the input data to be a sequence and do not generalize to unseen orders.

When a target variable depends on a population, the *rule combining* (Kersting and De Raedt 2001; Natarajan et al. 2010) approaches learn a distribution D for the target given only one object in the population, and then combine these distributions using an explicit aggregator (e.g., *mean* or *noisy-OR*). These models have three limitations compared to RelNNs: 1- they correspond to only two (non-flexible) layers, one for learning D and one for combining the predictions, 2- since D is learned separately for each object in the population, the interdependence among these objects is ignored, and 3- they rely on explicit aggregation function rather than learning the aggregator from the data.

Predicate invention approaches (Kemp et al. 2006; Kok and Domingos 2007) cluster the objects, their properties, and relationships such that the values of the target(s) depends on their clustering. Such a clustering can be implemented as one layer of a RelNN. Since latent variables in these works are probabilistic (rather than neural as in RelNNs), these works typically result in expensive models such that in practice clustering is limited to hard rather than soft clusters, and they have been applied to small domains with around $10k$ observations (which is far less than, say, the $1M$ observations in our datasets).

There are also several other works for learning deep networks from relational data, but they are all limited in terms of modelling power or scalability. The works in (Garcez, Broda, and Gabbay 2012; França, Zaverucha, and Garcez 2014; Serafini and Garcez 2016), for instance, propositionalize the data and miss the chance to learn specifically about objects. We show in our experiments that learning about objects substantially improves performance. Lodhi (2013) first learns features from relational data then feeds it into a standard neural network, thus learning features and the model independently. The same issue exists with methods such as DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) which first learn embeddings for nodes regardless of the prediction task, and then use the embeddings for making predictions. Šourek et al. (2015)'s models are the closest proposals to RelNNs, but RelNNs are more flexible in terms of adding new types of layers in a modular way. These works are also limited in one or more of the following ways: 1- the model is limited to only one input relation, 2- the structure of the model is highly dependent on the input data, 3- the model allows for only one hidden layer, 4- the model cannot learn hidden object properties through general rules, or 5- the model does not scale to large domains.

Conclusion

In this paper, we developed a deep neural model for learning from relational data. We showed that our model outperforms several existing models on three relational learning benchmarks. Our work can be advanced in several ways. The current implementation of our model is a fresh non-parallelized code. It could be speeded up by parallelization (e.g., by using TensorFlow (Abadi et al. 2016) as backbone), by compilation relational operations to lower-level languages (similar to (Kazemi and Poole 2016)), or by using advanced database query operations. Learning the structure of the RelNN and the WFs automatically from data (e.g., by extending the structure learning algorithm in (Fatemi, Kazemi, and Poole 2016)), developing and adding relational versions of regularization techniques such as (relational) dropout (Srivastava et al. 2014; Kazemi et al. 2017) and batch normalization (Ioffe and Szegedy 2015) are other directions for future research.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467.*

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *NIPS*, 2787–2795.

Das, R.; Neelakantan, A.; Belanger, D.; and McCallum, A. 2017. Chains of reasoning over entities, relations, and text using recurrent neural networks. *EACL.*

De Raedt, L.; Kersting, K.; Natarajan, S.; and Poole, D. 2016. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10(2):1–189.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7.

Fatemi, B.; Kazemi, S. M.; and Poole, D. 2016. A learning algorithm for relational logistic regression: Preliminary results. *arXiv preprint arXiv:1606.08531.*

França, M. V.; Zaverucha, G.; and Garcez, A. S. d. 2014. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning* 94(1):81–104.

Garcez, A. S. d.; Broda, K.; and Gabbay, D. M. 2012. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The weka data mining software: an update. *ACM SIGKDD explorations newsletter* 11(1).

Harper, M., and Konstan, J. 2015. The movielens datasets: History and context. *ACM TiiS* 5(4):19.

Huynh, T. N., and Mooney, R. J. 2008. Discriminative structure and parameter learning for Markov logic networks. In *Proc. of the ICML*.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Kazemi, S. M., and Poole, D. 2016. Knowledge compilation for lifted probabilistic inference: Compiling to a low-level language. In *KR*.

Kazemi, S. M.; Buchman, D.; Kersting, K.; Natarajan, S.; and Poole, D. 2014. Relational logistic regression. In *KR*.

Kazemi, S. M.; Fatemi, B.; Kim, A.; Peng, Z.; Tora, M. R.; Zeng, X.; Dirks, M.; and Poole, D. 2017. Comparing aggregators for relational probabilistic models. *UAI Workshop on Statistical Relational AI*.

Kemp, C.; Tenenbaum, J. B.; Griffiths, T. L.; Yamada, T.; and Ueda, N. 2006. Learning systems of concepts with an infinite relational model. In *AAAI*, volume 3, 5.

Kersting, K., and De Raedt, L. 2001. Adaptive bayesian logic programs. In *ICL*, 104–117. Springer.

Kok, S., and Domingos, P. 2007. Statistical predicate invention. In *ICML*, 433–440. ACM.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer* 42(8).

Lao, N., and Cohen, W. W. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine learning* 81(1):53–67.

Lao, N.; Mitchell, T.; and Cohen, W. W. 2011. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, 529–539.

Lodhi, H. 2013. Deep relational machines. In *Neural Information Processing*, 212–219. Springer.

Moore, J., and Neville, J. 2017. Deep collective inference. In *AAAI*, 2364–2372.

Natarajan, S.; Khot, T.; Lowd, D.; Tadepalli, P.; Kersting, K.; and Shavlik, J. 2010. Exploiting causal independence in markov logic networks: Combining undirected and directed models. In *Joint ECML and KDD*, 434–450.

Natarajan, S.; Khot, T.; Kersting, K.; Gutmann, B.; and Shavlik, J. 2012. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning* 86(1):25–56.

Neelakantan, A.; Roth, B.; and McCallum, A. 2015. Compositional vector space models for knowledge base inference. In *2015 aaai spring symposium series*.

Nguyen, D. Q. 2017. An overview of embedding models of entities and relationships for knowledge base completion. *arXiv preprint arXiv:1703.08098*.

Nickel, M.; Murphy, K.; Tresp, V.; and Gabrilovich, E. 2016. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE* 104(1):11–33.

Nickel, M.; Tresp, V.; and Kriegel, H.-P. 2012. Factorizing yago: scalable machine learning for linked data. In *World Wide Web*, 271–280. ACM.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *ICML*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proc. of the 20th ACM SIGKDD*, 701–710.

Pham, T.; Tran, T.; Phung, D. Q.; and Venkatesh, S. 2017. Column networks for collective classification. In *AAAI*, 2485–2491.

Poole, D.; Buchman, D.; Kazemi, S. M.; Kersting, K.; and Natarajan, S. 2014. Population size extrapolation in relational probabilistic modelling. In *SUM*.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62:107–136.

Rocktäschel, T., and Riedel, S. 2017. End-to-end differentiable proving. *arXiv preprint arXiv:1705.11040*.

Serafini, L., and Garcez, A. d. 2016. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*.

Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. 2013. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, 926–934.

Šourek, G.; Aschenbrenner, V.; Železny, F.; and Kuželka, O. 2015. Lifted relational neural networks. In *Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches-Volume 1583*, 52–60. CEUR-WS. org.

Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR* 15(1):1929–1958.

Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; and Bouchard, G. 2016. Complex embeddings for simple link prediction. In *ICML*, 2071–2080.

Uwents, W., and Blockeel, H. 2005. Classifying relational data with neural networks. In *ILP*, 384–396. Springer.

Wang, H.; Shi, X.; and Yeung, D.-Y. 2017. Relational deep learning: A deep latent variable model for link prediction. In *AAAI*.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable learning of logical rules for knowledge base reasoning. *arXiv preprint arXiv:1702.08367*.

Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R.; and Smola, A. 2017. Deep sets. *arXiv preprint arXiv:1703.06114*.