Neural Networks for Relational Data

Navdeep Kaur[1], Gautam Kunapuli[1], Saket Joshi[2], Kristian Kersting[3], and Sriraam Natarajan[1]

[1] The University of Texas at Dallas
{Navdeep.Kaur, Gautam.Kunapuli, Sriraam.Natarajan}@utdallas.edu
[2] Amazon Inc. {saketjoshi@gmail.com}
[3] TU Darmstadt, Germany {kersting@cs.tu-darmstadt.de}

Abstract. While deep networks have been enormously successful, they rely on flat-feature vector representations. Using them in structured domains requires significant feature engineering. Such domains rely on relational representations to capture complex relationships between entities and their attributes. Thus, we consider the problem of learning neural networks for relational data. We distinguish ourselves from current approaches that rely on expert hand-coded rules by learning higher-order random-walk features to capture local structural interactions and the resulting network architecture. We further exploit parameter tying, where instances of the same rule share parameters. Our experimental results demonstrate the effectiveness of the proposed approach over multiple neural net baselines as well as state-of-the-art relational models.

Keywords: neural networks · relational models

1 Introduction

Probabilistic Logic/Statistical Relational Models [1,3] allows them to model complex data structures such as graphs far more easily and interpretably than basic propositional representations. While expressive, these models do not incorporate or discover latent relationships between features as effectively as deep networks. There has been focus on achieving the dream team of symbolic and statistical learning methods such as *relational neural networks* [2, 7, 11, 15, 16]. While specific architectures differ, these methods generally employ *an expert* or Inductive Logic Programming (ILP, [10]) to identify domain structure/rules which are then instantiated to learn a neural network. We improve upon these methods in two ways: (1) we employ a rule learner to automatically extract *interpretable* rules that are then employed as hidden layer of the neural network; (2) we exploit *parameter tying* similar to SRL models [14] that allow multiple instances of the same rule share the same parameter. These two extensions significantly improve the adaptation of neural networks (NNs) for relational data.

We employ *Relational Random Walks* [9] to extract relational rules from a database, which are then used as the first layer of the NN. These random walks have the advantages of being learned from data (as against time-consumingly hand-coded features) and interpretable (they are rules in a database schema). Given evidence (facts), our network ensure two key features. First, relational

random walks are learned and instantiated (grounded); parameter tying ensures that groundings of the same random walk share the same parameters resulting in far fewer learnable network parameters. Next, for combining outputs from different groundings of the *same clause*, we employ combination functions [4,14]. Finally, once the network weights are appropriately constrained by parameter tying, they can be learned using standard techniques such as backpropagation.

We make the following contributions: (1) we learn a NN that can be fully trained from data and with no significant engineering, unlike previous approaches; (2) we combine the successful paradigms of relational random walks and parameter tying allowing the resulting NN to faithfully model relational data while being fully learnable; (3) we evaluate the approach and demonstrate its efficacy.

2 Related Work

Our work is closest to Lifted Relational Neural Networks (LRNN, [15]) due to Šourek et al. LRNN uses expert hand-crafted rules, which are then instantiated and rolled out as a ground network. While Šourek et al., exploit tied parameters across facts, we share parameters *across multiple instances* of the same rule. LRNN supports weighted facts due to the fuzzy notion that they adapt; we take a more standard approach with Boolean facts. Finally, while the previous difference appears to be limiting, in our case it leads to a *reduction in the number of network weights*. Ŝourek et al., extended their work to learn network structure using predicate invention [16]; our work learns relational random walks as rules for the network structure. As we show in our experiments, NNs can easily handle large numbers of random walks as weakly predictive intermediate layers capturing local features. This allows for learning a more robust model than the induced rules, which take a more global view of the domain.

Another recent approach is due to Kazemi and Poole [7], who proposed a relational neural network by adding hidden layers to their Relational Logistic Regression [6] model. A key limitation of their work is that they are restricted to unary relation predictions. Other recent approaches such as CILP++ [2] and Deep Relational Machines [11] incorporate relational information as network layers. However, such models propositionalize relational data into flat-feature vector while we learn a lifted model.

3 Neural Networks with Relational Parameter Tying

A relational neural network $\mathcal{N}$ is a set of M weighted rules $\{\mathtt{R_j}, w_j)\}_{j=1}^{M}$. Relational rules are conjunctions of the form $\mathtt{h} \Leftarrow \mathtt{b_1} \wedge \ldots \wedge \mathtt{b}_\ell$, where the head $\mathtt{h}$ is the target of prediction and the body $\mathtt{b_1} \wedge \ldots \wedge \mathtt{b}_\ell$ corresponds to conditions that make up the rule. We are given evidence: atomic facts $\mathcal{F}$ and labeled relational examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$. We seek to learn a relational neural network $\mathcal{N} \equiv \{\mathtt{R_j}, w_j)\}_{j=1}^{M}$ to predict a $\mathtt{Target}$ relation: $y = \mathtt{Target}(\mathbf{x})$. This consists of two steps: structure learning, to learn the architecture of $\mathcal{N}$ and parameter learning, to identify (tied) network parameters of $\mathcal{N}$.

3.1 Generating Lifted Random Walks

The architecture is determined by the *set of induced clauses from the domain*. While previous approaches employed carefully hand-crafted rules, we use relational random walks to define the network architecture and local qualitative structure of the domain. Relational data is often represented using a lifted graph, which defines the domain's schema; in such a representation, a relation $\texttt{Predicate}(\texttt{Type}_1, \texttt{Type}_2)$ can be understood as a predicate edge between two type nodes: $\texttt{Type}_1 \xrightarrow{\texttt{Predicate}} \texttt{Type}_2$. A relational random walk through a graph is a chain of such edges corresponding to a conjunction of predicates.

For a random walk to be semantically sound, the input type (argument domain) of the $(i+1)$-th predicate must be the same as the output type (argument range) of the i-th predicate. For example, in a movie domain, the body of the rule $\texttt{ActedIn}(\texttt{P}_1, \texttt{G}_1) \wedge \texttt{SameGenre}(\texttt{G}_1, \texttt{G}_2) \wedge \texttt{ActedIn}^{-1}(\texttt{G}_2, \texttt{P}_2) \wedge \texttt{SamePerson}(\texttt{P}_2, \texttt{P}_3) \Rightarrow \texttt{WorkedUnder}(\texttt{P}_1, \texttt{P}_3)$ is a lifted random walk $\texttt{P}_1 \xrightarrow{\texttt{ActedIn}} \texttt{G}_1 \xrightarrow{\texttt{SameGenre}} \texttt{G}_2 \xrightarrow{\texttt{ActedIn}^{-1}} \texttt{P}_2 \xrightarrow{\texttt{SamePerson}} \texttt{P}_3$, between two entities $\texttt{P}_1 \rightarrow \texttt{P}_3$ in the target predicate, $\texttt{WorkedUnder}(\texttt{P}_1, \texttt{P}_3)$. This walk contains an *inverse predicate* $\texttt{ActedIn}^{-1}$, which is distinct from $\texttt{ActedIn}$ (its arguments are reversed).

We use path-constrained random walks [9] approach to generate M lifted random walks $\texttt{R}_\texttt{j}$, $j = 1, \ldots, M$. These random walks form the backbone of the lifted neural network, as they are templates for various feature combinations in the domain. They can also be interpreted as *domain rules* as they impart localized structure, or a qualitative description of the domain. When these lifted random walks have weights associated with them, we are then able to endow them with a quantitative influence on the target. A key component of network instantiation with these rules is *rule-based parameter tying*, which reduces the number of learnable parameters significantly, while still effectively maintaining the quantitative influences as described by the relational random walks.

3.2 Network Instantiation

The relational random walks ($\texttt{R}_\texttt{j}$) generated above are the relational features of the lifted relational NN, $\mathcal{N}$. Our goal is to unroll and ground the network with several intermediate layers that capture the relationships expressed by these random walks. A key difference in network construction between our proposed work and recent approaches [15] is that *we do not perform an exhaustive grounding* to generate all possible instances before constructing the network. Instead, we only ground as needed leading to a much more compact network (cf. Figure 1).

Output Layer: For the $\texttt{Target}$, which is also the head $\texttt{h}$ in all the rules $\texttt{R}_\texttt{j}$, introduce an output neuron called the *target neuron*, $A_\texttt{h}$. With one-hot encoding of the target labels, this architecture can handle multi-class problems. The target neuron uses the *softmax activation function*. Without loss of generality, we describe the rest of the network unrolling assuming a single output neuron.

Combining Rules Layer: The target neuron is connected to M *lifted rule neurons*, each corresponding to one of the lifted relational random walks,

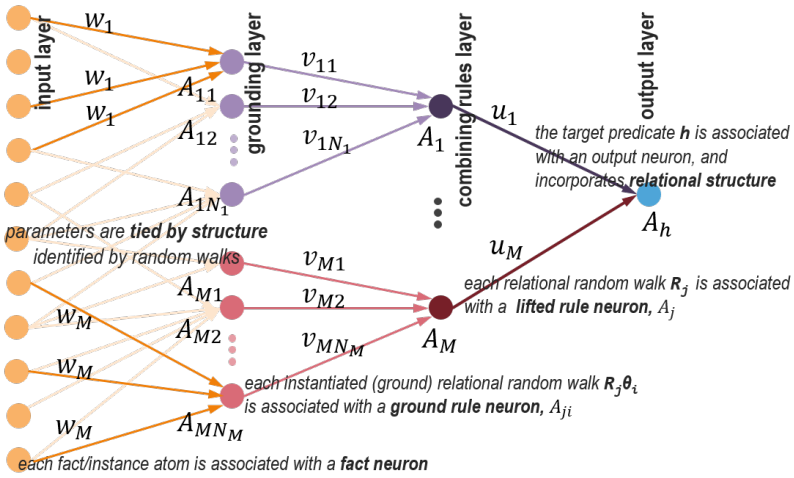

Fig. 1. The relational neural network is unrolled in three stages, ensuring that the output is a function of facts through two hidden layers: the combining rules layer (with lifted random walks) and the grounding layer (with instantiated random walks). Weights are tied between the input and grounding layers based on which fact/feature ultimately contributes to which rule in the combining rules layer.

$(\mathtt{R_j}, w_j)$. Each rule $\mathtt{R_j}$ is a conjunction of predicates defined by random walks:

$$\mathtt{Q_1^j}(\mathtt{X}, \cdot) \wedge \ldots \wedge \mathtt{Q_L^j}(\cdot, \mathtt{Z}) \Rightarrow \mathtt{Target}(\mathtt{X}, \mathtt{Z}),\ j = 1, \ldots, M,$$

and corresponds to the lifted rule neuron A_j. This layer of neurons is fully connected to the output layer to ensure that all the lifted random walks (that capture the domain structure) influence the output. The extent of their influence is determined by learnable weights, u_j between A_j and the output neuron A_h.

In Fig. 1, we see that the rule neuron A_j is connected to the neurons A_{ji}; these neurons correspond to N_j *instantiations* of the random-walk $\mathtt{R_j}$. The lifted rule neuron A_j aims to *combine the influence of the groundings* of the random-walk feature $\mathtt{R_j}$ that are true in the evidence. Thus, each lifted rule neuron can also be viewed as a *rule combination neuron*. The activation function of a rule combination neuron can be any *aggregator or combining rule* [14]. This can include *value aggregators* such as **weighted mean, max** or *distribution aggregators* (if inputs to the this layer are probabilities) such as **Noisy-Or**. For instance, combining rule instantiations $\mathtt{out}(A_{ji})$ with a weighted mean will require learning v_{ji}, with the nodes using unit functions for activation. This layer is more general and subsumes LRNN [15], which uses a max combination layer.

Grounding Layer: For each ground random walk $\mathtt{R_j}\boldsymbol{\theta}_i$, $i = 1, \ldots, N_j$, we introduce a *ground rule neuron*, A_{ji}. This ground rule neuron represents the i-th instantiation (grounding) of the body of the j-th rule, $\mathtt{R_j}\boldsymbol{\theta_i}$: $\mathtt{Q_1^j}\boldsymbol{\theta_i} \wedge \ldots \wedge \mathtt{Q_\ell^j}\boldsymbol{\theta_i}$. The activation function of a ground rule neuron is a *logical AND* ($\wedge$); it is only activated when all its constituent inputs are true. This requires all the constituent facts $\mathtt{Q_1^j}\boldsymbol{\theta_i}, \ldots, \mathtt{Q_\ell^j}\boldsymbol{\theta_i}$ to be in the evidence. Thus, the (j, i)-th ground rule neuron is connected to all the *fact neurons* that appear in its corresponding instantiated

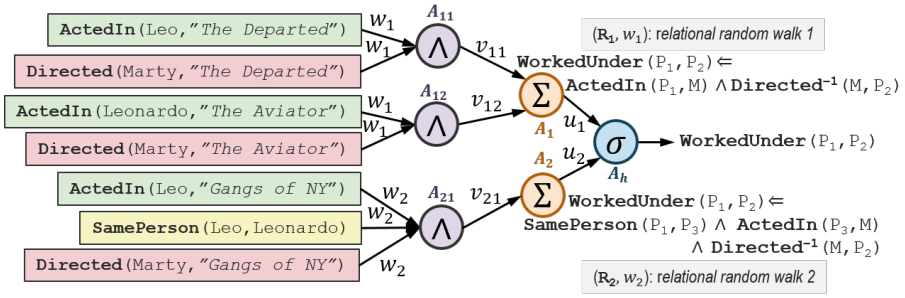

Fig. 2. Example: unrolling the network with relational parameter tying.

rule body. A key novelty is relational parameter tying: the weights of connections between the fact and grounding layers are *tied* by the rule these facts appear in together. This is described in detail further below.

Input Layer: Each grounded predicate is a *fact*, that is $\mathtt{Q}^{\mathtt{j}}_{\mathtt{k}}\boldsymbol{\theta}_{\mathtt{i}} \in \mathcal{F}$. For each such instantiated fact, we create a *fact neuron* A_f, ensuring that each unique fact in evidence has only one single neuron associated with it. Every example is a collection of facts, that is, example $\mathbf{x}_i \equiv \mathcal{F}_i \subset \mathcal{F}$. Thus, an example is input into the system by simply activating its constituent facts in the input layer.

Relational Parameter Tying: We employ *rule-based parameter tying* for the weights between the grounding layer and the input/facts layer. Parameter tying ensures that instances corresponding to an example all share the same weight w_j if they occur in the same lifted rule $\mathtt{R_j}$. The shared weights w_j are propagated through the network in a bottom-up fashion, ensuring that weights in the succeeding hidden layers are influenced by them. Our approach to parameter tying is in sharp contrast to LRNN, where weights between the output and combining rules layers are learned. This approach also differs from our previous work [5] where we used relational random walks as features for an RBM. The relational RBM formulation has significantly more edges, and many more parameters to optimize during learning as its architecture is a bipartite graph.

We illustrate network construction with an example: two lifted random walks $(\mathtt{R_1}, w_1)$ and $(\mathtt{R_2}, w_2)$ for the target predicate $\mathtt{WorkedUnder}(\mathtt{P_1}, \mathtt{P_2})$ (Fig. 2):

$$\mathtt{WorkedUnder}(\mathtt{P_1}, \mathtt{P_2}) \Leftarrow \mathtt{ActedIn}(\mathtt{P_1}, \mathtt{M}) \wedge \mathtt{Directed}^{-1}(\mathtt{M}, \mathtt{P_2}),$$
$$\mathtt{WorkedUnder}(\mathtt{P_1}, \mathtt{P_2}) \Leftarrow \mathtt{SamePerson}(\mathtt{P_1}, \mathtt{P_3}) \wedge \mathtt{ActedIn}(\mathtt{P_3}, \mathtt{M}) \wedge \mathtt{Directed}^{-1}(\mathtt{M}, \mathtt{P_2}).$$

The **output layer** consists of a single neuron A_h corresponding to $\mathtt{WorkedUnder}$. The **lifted rule layer** has two lifted rule nodes A_1 corresponding to rule $\mathtt{R_1}$ and A_2 corresponding to rule $\mathtt{R_2}$. These rule nodes combine inputs corresponding to instantiations that are true in the evidence. The network is unrolled based on the specific training example, for instance: $\mathtt{WorkedUnder}(\mathtt{Leo}, \mathtt{Marty})$. For this example, the rule $\mathtt{R_1}$ has two instantiations that are true in the evidence. Then, we introduce a ground rule node for each such instantiation:

$$A_{11} : \mathtt{Act}(\mathtt{Leo}, \text{"}\mathtt{TheDeparted}\text{"}) \wedge \mathtt{Dir}^{-1}(\text{"}\mathtt{TheDeparted}\text{"}, \mathtt{Marty}),$$
$$A_{12} : \mathtt{Act}(\mathtt{Leo}, \text{"}\mathtt{TheAviator}\text{"}) \wedge \mathtt{Dir}^{-1}(\text{"}\mathtt{TheAviator}\text{"}, \mathtt{Marty}).$$

The rule $\mathtt{R_2}$ has only one instantiation, and consequently only one node:

$$A_{21} : \mathtt{SamPer}(\mathtt{Leo}, \mathtt{Leonardo}) \wedge \mathtt{Act}(\mathtt{Leo}, \text{"}\mathtt{TheDeparted}\text{"}) \wedge \mathtt{Dir}^{-1}(\text{"}\mathtt{TheDeparted}\text{"}, \mathtt{Marty}).$$

The **grounding layer** consists of ground rule nodes corresponding to instantiations of rules that are true in the evidence. The edges $A_{ji} \rightarrow A_j$ have weights v_{ji} that depend on the combining rule implemented in A_j. In this example, the combining rule is *average*, so we have $v_{11} = v_{12} = \frac{1}{2}$ and $v_{21} = 1$. The **input layer** consists of atomic fact in evidence: $f \in \mathcal{F}$. The fact nodes $\mathtt{ActedIn}(\mathtt{Leo}, \text{"}\mathtt{TheAviator}\text{"})$ and $\mathtt{Directed}^{-1}(\text{"}\mathtt{TheAviator}\text{"}, \mathtt{Marty})$ appear in the grounding $\mathtt{R_1}\boldsymbol{\theta}_2$ and are connected to the corresponding ground rule neuron A_{12}. Finally, parameters are tied between the facts layer and the grounding layer. This ensures that all facts that ultimately contribute to a rule are pooled together, which increases the influence of the rule during weight learning. This ensures that a rule that holds strongly in the evidence gets a higher weight.

4 Experiments

We aim to answer the following questions about our method, *Neural Networks with Relational Parameter Tying* (NNRPT):[4] **Q1:** How does NNRPT compare to the state-of-the-art SRL models ? **Q2:** How does NNRPT compare to propositionalization models ? **Q3:** How does NNRPT compare to relational neural networks in literature?

Table 1. Data sets used in our experiments. The last column is the number of sampled groundings of random walks per example for NNRPT.

Domain	Target	#Facts	#Pos	#Neg	#RW	#Samp/RW
Uw-Cse	advisedBy	2817	90	180	2500	1000
Mutagenesis	MoleAtm	29986	1000	2000	100	100
Cora	SameVenue	31086	2331	4662	100	100
Imdb	WorkedUnder	914	305	710	80	-
Sports	TeamPlaysSport	7824	200	400	200	100

To answer **Q1**, we compare NNRPT with state-of-the-art relational gradient-boosting methods, RDN-Boost [13], MLN-Boost [8], and relational RBMs (RRBM-E, RRBM-C, [5]). As the random walks chain binary predicates in NNRPT, we convert unary and ternary predicates into binary predicates for all data sets. We use these binary predicates across *all our baselines*. We run RDN-Boost and MLN-Boost with their default settings and learn 20 trees for each model. We train RRBM-E and RRBM-C according to the settings recommended in [5]. For NNRPT, we generate random walks by considering each predicate and its inverse to be two distinct predicates. Also, we avoid loops in the random walks by enforcing sanity constraints on the random walk generation (Table 1). When *exhaustive grounding becomes prohibitively expensive*, we *sample groundings* for each random walk for large data sets. For all experiments, we set the positive to negative ratio to be

[4] https://github.com/navdeepkjohal/NNRPT

Table 2. Comparison with different SRL algorithms.

Data Set	Measure	RDN-Boost	MLN-Boost	RRBM-E	RRBM-C	NNRPT
Uw-Cse	AUC-ROC	0.973±0.014	0.968±0.014	0.975±0.013	0.968±0.011	0.959±0.024
	AUC-PR	0.931±0.036	0.916±0.035	0.923±0.056	0.924±0.040	0.896±0.063
Imdb	AUC-ROC	0.955±0.046	0.944±0.070	1.000±0.000	0.997±0.006	0.984±0.025
	AUC-PR	0.863±0.112	0.839±0.169	1.000±0.000	0.992±0.017	0.951±0.082
Cora	AUC-ROC	0.895±0.183	0.835±0.035	0.984±0.009	0.867±0.041	0.952±0.043
	AUC-PR	0.833±0.259	0.799±0.034	0.948±0.042	0.825±0.050	0.899±0.070
Mutag.	AUC-ROC	0.999±0.000	0.999±0.000	0.999±0.000	0.998±0.001	0.981±0.024
	AUC-PR	0.999±0.000	0.999±0.000	0.999±0.000	0.997±0.002	0.970±0.039
Sports	AUC-ROC	0.801±0.026	0.806±0.016	0.760±0.016	0.656±0.071	0.780±0.026
	AUC-PR	0.670±0.028	0.652±0.032	0.634±0.020	0.648±0.085	0.668±0.070

Table 3. Comparison of NNRPT with propositionalization-based approaches.

Data Set	Measure	BCP-RBM	BCP-NN	NNRPT
Uw-Cse	AUC-ROC	0.951±0.041	0.868±0.053	0.959±0.024
	AUC-PR	0.860±0.114	0.869±0.033	0.896±0.063
Imdb	AUC-ROC	0.780±0.164	0.540±0.152	0.984±0.025
	AUC-PR	0.367±0.139	0.536±0.231	0.951±0.082
Cora	AUC-ROC	0.801±0.017	0.670±0.064	0.952±0.043
	AUC-PR	0.647±0.050	0.658±0.064	0.899±0.070
Mutag.	AUC-ROC	0.991±0.003	0.945±0.019	0.981±0.024
	AUC-PR	0.995±0.001	0.973±0.012	0.970±0.039
Sports	AUC-ROC	0.664±0.021	0.543±0.037	0.780±0.026
	AUC-PR	0.532±0.041	0.499±0.065	0.668±0.070

$1:2$ for training, set combination function to "average" and perform 5-fold cross validation. For NNRPT, we set the learning rate to be 0.05, batch size to 1, and number of epochs to 1. We train our model with L_1-regularized AdaGrad.

To answer **Q2**, we generated flat feature vectors by Bottom Clause Propositionalization (BCP, [2]), according to which one bottom clause is generated for each example. BCP considers each predicate in the body of the bottom clause as a unique feature when it propositionalizes bottom clauses to flat feature vector. We use PROGOL [12] to generate bottom clauses. After propositionalization, we train two models: a propositionalized RBM (BCP-RBM) and a propositionalized NN (BCP-NN). The NN has two hidden layers, which makes BCP-NN model a modified version of CILP++ [2] that has one hidden layer. Hyper-parameters of both models were optimized by line search on a validation set. To answer **Q3**, we compare NNRPT with LRNN [15]. To ensure fairness, we perform structure learning by using PROGOL and input the *same clauses* to both LRNN and NNRPT. PROGOL learned 4 clauses for Cora, 8 clauses for Imdb, 3 clauses for Sports, 10 clauses for Uw-Cse and 11 clauses for Mutagenesis in our experiment.

Table 4. Comparison of NNRPT and LRNN on AUC-ROC and AUC-PR on different data sets. Both the models were provided clauses learnt by PROGOL, [12].

Model	Measure	Uw-Cse	Imdb	Cora	Mutagen.	Sports
LRNN	AUC-ROC	0.923±0.027	0.995±0.004	0.503±0.003	0.500±0.000	0.741±0.016
	AUC-PR	0.826±0.056	0.985±0.013	0.356±0.006	0.335±0.000	0.527±0.036
NNRPT	AUC-ROC	0.700±0.186	0.997±0.007	0.968±0.022	0.532±0.019	0.657±0.014
	AUC-PR	0.910±0.072	0.992±0.017	0.943±0.032	0.412±0.032	0.658±0.056

Table 2 compares NNRPT to state-of-the-art SRL methods. NNRPT is significantly better than RRBM for Cora and Sports, and performs comparably on other data sets. It also performs better than MLN-Boost, RDN-Boost on Imdb and Cora, and comparably on other data sets. Broadly, **Q1** can be answered affirmatively: NNRPT performs comparably to or better than state-of-the-art SRL.

Table 3 compares NNRPT with two propositionalization models: BCP-RBM and BCP-NN. NNRPT performs better than BCP-RBM on all data sets except Mutagenesis, where the two are comparable. NNRPT performs better than BCP-NN on all data sets. It should be noted that BCP feature generation sometimes introduces a large positive-to-negative example skew (for example, in the Imdb data set), which can sometimes gravely affect the performance of propositional models. This emphasizes the need for models, like ours, that can handle relational data without propositionalization. **Q2** can now be answered affirmatively: that NNRPT performs better than propositionalization models.

Table 4 compares NNRPT and LRNN when both use clauses learned by PROGOL. NNRPT performs better on Uw-Cse, Sports evaluated using AUC-PR. This result is especially significant because these data sets are considerably skewed. NNRPT also outperforms LRNN on Cora and Mutagenesis. The reason for this big performance gap between the two models on Cora is likely because LRNN could not build effective models with the fewer number of clauses typically learned by PROGOL (four, here). In contrast, even with very few clauses, NNRPT outperforms LRNN. This helps us answer **Q3**, affirmatively: NNRPT offers many advantages over state-of-the-art relational neural networks.

Our experiments show the benefits of parameter tying as well as the expressivity of relational random walks in tightly integrating with a neural network model across a variety of domains and settings. The key strengths of NNRPT are that it can (1) efficiently incorporate a large number of relational features, (2) capture local qualitative structure through relational random walk features, (3) tie feature weights to capture global quantitative influences.

5 Conclusion and Future Work

We considered the problem of learning neural networks from relational data. Our proposed architecture exploits parameter tying: instances of the same rule share the same parameters for the same training example. In addition, we explored relational random walks as relational features for training these neural nets. Fur-

ther experiments on larger data sets could yield insights into the scalability of this approach. Integration with an approximate-counting method could potentially reduce the training time. Finally, understanding the use of such random-walk-based NN as a function approximator can allow for efficient and interpretable learning in relational domains with minimal feature engineering.

Acknowledgements: SN, GK & NK gratefully acknowledge AFOSR award FA9550-18-1-0462. KK acknowledges the support of the RMU project DeCoDeML. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the AFOSR, DeCoDeML or the US government.

References

1. De Raedt, L., Kersting, K., Natarajan, S., Poole, D.: Statistical Relational Artificial Intelligence: Logic, Probability, and Computation. Morgan & Claypool (2016)
2. França, M.V.M., Zaverucha, G., d'Avila Garcez, A.S.: Fast relational learning using bottom clause propositionalization with artificial neural networks. MLJ (2014)
3. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press (2007)
4. Jaeger, M.: Parameter learning for relational bayesian networks. In: ICML (2007)
5. Kaur, N., Kunapuli, G., Khot, T., Kersting, K., Cohen, W., Natarajan, S.: Relational restricted Boltzmann machines: A probabilistic logic learning approach. In: ILP (2017)
6. Kazemi, S.M., Buchman, D., Kersting, K., Natarajan, S., Poole, D.: Relational logistic regression. In: KR (2014)
7. Kazemi, S.M., Poole, D.: RelNN: A deep neural model for relational learning. In: AAAI (2018)
8. Khot, T., Natarajan, S., Kersting, K., Shavlik, J.: Learning Markov logic networks via functional gradient boosting. In: ICDM (2011)
9. Lao, N., Cohen, W.: Relational retrieval using a combination of path-constrained random walks. JMLR (2010)
10. Lavrac, N., Džeroski, v.: Inductive Logic Programming: Techniques and Applications. Prentice Hall (1993)
11. Lodhi, H.: Deep relational machines. In: ICONIP (2013)
12. Muggleton, S.: Inverse entailment and Progol. New Generation Computing (1995)
13. Natarajan, S., Khot, T., Kersting, K., Guttmann, B., Shavlik, J.: Gradient-based boosting for statistical relational learning: Relational dependency network case. MLJ (2012)
14. Natarajan, S., Tadepalli, P., Dietterich, T.G., Fern, A.: Learning first-order probabilistic models with combining rules. ANN MATH ARTIF INTEL (2008)
15. Šourek, G., Aschenbrenner, V., Železny, F., Kuželka, O.: Lifted relational neural networks. In: NeurIPS Workshop (2015)
16. Šourek, G., Svatoš, M., Železný, F., Schockaert, S., Kuželka, O.: Stacked structure learning for lifted relational neural networks. In: ILP (2017)