GO FOR A WALK AND ARRIVE AT THE ANSWER: REASONING OVER PATHS IN KNOWLEDGE BASES USING REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Knowledge bases (KB), both automatically and manually constructed, are often incomplete — many valid facts can be inferred from the KB by synthesizing existing information. A popular approach to KB completion is to infer new relations by combinatory reasoning over the information found along other paths connecting a pair of entities. Given the enormous size of KBs and the exponential number of paths, previous path-based models have considered only the problem of predicting a missing relation given two entities, or evaluating the truth of a proposed triple. Additionally, these methods have traditionally used random paths between fixed entity pairs or more recently learned to pick paths between them. We propose a new algorithm, MINERVA, which addresses the much more difficult and practical task of answering questions where the relation is known, but only one entity. Since random walks are impractical in a setting with combinatorially many destinations from a start node, we present a neural reinforcement learning approach which learns how to navigate the graph conditioned on the input query to find predictive paths. Empirically, this approach obtains state-of-the-art results on several datasets, significantly outperforming prior methods.

1 INTRODUCTION

Automated reasoning, the ability of computing systems to make new inferences from observed evidence, has been a long standing goal of artificial intelligence. We are interested in automated reasoning on large knowledge bases (KB) with rich and diverse semantics (Suchanek et al., 2007; Bollacker et al., 2008; Carlson et al., 2010). KBs are highly incomplete (Min et al., 2013), and facts not directly stored in a KB can often be inferred from those that are, creating exciting opportunities and challenges for automated reasoning. For example, consider the small knowledge graph in figure 1. We can infer the (unobserved fact) home stadium of Colin Kaepernick from the following reasoning *path*: Colin Kaepernick $\rightarrow$ PlaysInTeam $\rightarrow$ 49ers $\rightarrow$ TeamHomeStadium $\rightarrow$ Levi's Stadium. Our goal is to automatically learn such reasoning paths in KBs. We frame the learning problem as one of query answering, that is to say, answering questions of the form (Colin Kaepernick, PlaysInLeague, ?).

From its early days, the focus of automated reasoning approaches has been to build systems which can learn crisp symbolic logical rules (McCarthy, 1960; Nilsson, 1991). Symbolic representations have also been integrated with machine learning especially in statistical relational learning (Muggleton et al., 1992; Getoor & Taskar, 2007; Kok & Domingos, 2007; Lao et al., 2011), but due to poor generalization performance, these approaches have largely been superceded by distributed vector representations. Learning embedding of entities and relations using tensor factorization or neural methods has been a popular approach (Nickel et al., 2011; Bordes et al., 2013; Socher et al., 2013; inter alia), but these methods cannot capture chains of reasoning expressed by KB paths. Neural multi-hop models (Neelakantan et al., 2015; Guu et al., 2015; Toutanova et al., 2016) address the aforementioned problems to some extent by operating on KB paths in vector space. However, these models take as input a set of paths which are gathered by performing random walks *independent* of the query relation. Additionally, models such as Neelakantan et al. (2015); Das et al. (2017) use the same set of initially collected paths to answer a diverse set of query types (e.g. MarriedTo, Nationality, WorksIn etc.).

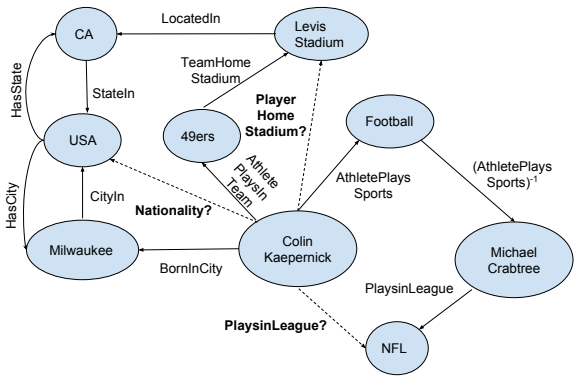

Figure 1: A small fragment of a knowledge base represented as a knowledge graph. Solid edges are observed and dashed edges are part of queries. Note how each query (e.g. Nationality, PlaysInLeague, PLayerHomeStadium) can be answered by traversing the graph via "logical" paths between entity 'Colin Kaepernick' and the corresponding answer.

This paper presents a method for efficiently searching the graph for answer-providing paths using reinforcement learning (RL) conditioned on the input question, eliminating any need for precomputed paths. Given a massive knowledge graph, we learn a policy, which, given the query $(\mathit{entity}_1, \mathit{relation}, ?)$, starts from entity_1 and learns to walk to the answer node by choosing to take a labeled relation edge at each step, *conditioning* on the query relation and entire path history. This formulates the query-answering task as a reinforcement learning (RL) problem where the goal is to take an optimal sequence of decisions (choices of relation edges) to maximize the expected reward (reaching the correct answer node). We call the RL agent MINERVA for "Meandering In Networks of Entities to Reach Verisimilar Answers."

Our RL-based formulation has many desirable properties. First, MINERVA has the built-in flexibility to take paths of variable length, which is important for answering harder questions that require complex chains of reasoning (Shen et al., 2017). Secondly, MINERVA needs *no pretraining* and trains on the knowledge graph from scratch with reinforcement learning; no other supervision or fine-tuning is required representing a significant advance over prior applications of RL in NLP. Third, our path-based approach is computationally efficient, since by searching in a small neighborhood around the query entity it avoids ranking all entities in the KB as in prior work. Finally, the reasoning paths found by our agent automatically form an interpretable provenance for its predictions.

The main contributions of the paper are: (a) We present agent MINERVA, which learns to do query answering by walking on a knowledge graph conditioned on an input query, stopping when it reaches the answer node. The agent is trained using reinforcement learning, specifically policy gradients (§ 2). (b) We evaluate MINERVA on several benchmark datasets and compare favorably to Neural Theorem Provers (NTP) (Rocktäschel & Riedel, 2017) and Neural LP (Yang et al., 2017), which do logical rule learning in KBs, and also state-of-the-art embedding based methods such as DistMult (Yang et al., 2015) and ComplEx (Trouillon et al., 2016). (c) We also extend MINERVA to handle partially structured natural language queries and test it on the WikiMovies dataset (§ 4.3) (Miller et al., 2016).

We also compare to DeepPath (Xiong et al., 2017) which uses reinforcement learning to pick paths between entity pairs. The main difference is that the state of their RL agent includes the answer entity since it is designed for the simpler task of predicting if a fact is true or not. As such their method cannot be applied directly to our more challenging query answering task where the second entity is unknown and must be inferred. Nevertheless, MINERVA outperforms DeepPath on their benchmark NELL-995 dataset when compared in their experimental setting (§ 4.1).

2 TASK AND MODEL

We formally define the task of query answering in a KB. Let $\mathcal{E}$ denote the set of entities and $\mathcal{R}$ be the set of binary relations. Then a KB is a collection of facts stored as triplets (e_1, r, e_2) where $e_1, e_2 \in \mathcal{E}$ and $r \in \mathcal{R}$. Query answering seeks to answer questions of the form $(e_1, r, ?)$, e.g. Toronto, locatedIn, ?. We would also like to clearly point out the difference between query answering and the task of fact prediction. Fact prediction involves predicting if a fact is true or not, e.g. (Toronto, locatedIn, Canada)?. This task is easier than predicting the correct entity as the answer in query answering since the latter require finding the answer entity among many possible entities.

Next we describe how we reduce the problem of query answering in a KB to a finite horizon sequential decision making problem and solve it using reinforcement learning. We begin by representing the environment as a deterministic Markov decision process on a knowledge graph $\mathcal{G}$ derived from

the KB (§2.1). Our RL agent is given an input query of the form $\left(\mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}}, ?\right)$. Starting from vertex corresponding to e_{1q} in the knowledge graph $\mathcal{G}$, the agent learns to traverse the environment/graph to mine the answer and stop when it determines the answer (§ 2.2). The agent is trained using policy gradient more specifically by REINFORCE (Williams, 1992) with control variates (§ 2.3). Let us begin by describing the environment.

2.1 Environment - States, Actions, Transitions and Rewards

Our environment is a finite horizon, deterministic and partially observed Markov decision process that lies on a knowledge graph derived from the KB. Recall that a KB is collection of facts stored as triplets $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2)$ where $\mathrm{e}_1, \mathrm{e}_2 \in \mathcal{E}$ and $r \in \mathcal{R}$. From the KB, a knowledge graph $\mathcal{G}$ can be constructed where the entities $\mathrm{e}_1, \mathrm{e}_2$ are represented as the nodes and relation r as labeled edge between them. Formally, a knowledge graph is a directed labeled multigraph $\mathcal{G} = (V, E, \mathcal{R})$, where V and E denote the vertices and edges of the graph respectively. Note that $V = \mathcal{E}$ and $E \subseteq V \times \mathcal{R} \times V$. Also, following previous approaches (Bordes et al., 2013; Neelakantan et al., 2015; Xiong et al., 2017), we add the inverse relation of every edge, i.e. for an edge $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2) \in E$, we add the edge $(\mathrm{e}_2, \mathrm{r}^{-1}, \mathrm{e}_1)$ to the graph. (If the set of binary relations $\mathcal{R}$ does not contain the inverse relation r^{-1}, it is added to $\mathcal{R}$ as well.) On this graph we will now specify a deterministic partially observable Markov decision process, which is a 5-tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \delta, R)$, each of which we elaborate below.

States. The state space $\mathcal{S}$ consists of all possible query-answers cartesian product with the set of entities. Intuitively, we want a state to encode the query $(\mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}})$, the answer $(\mathrm{e}_{2\mathrm{q}})$, and a location of exploration e_{t} (current node of the entity). Thus overall a state $S \in \mathcal{S}$ is represented by $S = (\mathrm{e}_{\mathrm{t}}, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}}, \mathrm{e}_{2\mathrm{q}})$ and the state space consists of all valid combinations.

Observations. The complete state of the environment is not observable, but only its current location of exploration and query can be observed but not the answer, i.e. only $(\mathrm{e}_{\mathrm{t}}, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}})$ is observed. Formally the observation function $\mathcal{O} : \mathcal{S} \to V \times V \times \mathcal{R}$ is defined as $\mathcal{O}(s = (\mathrm{e}_{\mathrm{t}}, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}}, \mathrm{e}_{2\mathrm{q}})) = (\mathrm{e}_{\mathrm{t}}, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}})$.

Actions. The set of possible actions $\mathcal{A}_S$ from a state $S = (\mathrm{e}_{\mathrm{t}}, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}}, \mathrm{e}_{2\mathrm{q}})$ consists of all outgoing edges of the vertex e_{t} in $\mathcal{G}$. Formally $\mathcal{A}_S = \{(\mathrm{e}_{\mathrm{t}}, r, v) \in E : S = (\mathrm{e}_{\mathrm{t}}, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}}, \mathrm{e}_{2\mathrm{q}}), r \in \mathcal{R}, v \in V\} \cup \{(s, \varnothing, s)\}$. Basically, this means an agent at each state has option to select which outgoing edge it wishes to take having the knowledge of the label of the edge r and destination vertex v.

During implementation, we unroll the computation graph up to a fixed number of time steps T. We augment each node with a special action called 'NO_OP' which goes from a node to itself. Some questions are easier to answer and needs lesser steps of reasoning than others. This design decision allows the agent to remain at a node for any number of time steps. This is especially helpful when the agent has managed to reach a correct answer at a time step $t <$ T and can continue to stay at the 'answer node' for the rest of the time steps. Alternatively, we could have allowed the agent to take a special 'STOP' action, but we found the current setup to work sufficiently well. As mentioned before, we also add the inverse relation of a triple, i.e. for the triple (e_1, r, e_2), we add the triple (e_2, r^{-1}, e_1) to the graph. We found this important because this actually equips our agent to *undo* a potentially wrong decision as it can retract back to the current node in the next step.

Transition. The environment evolves deterministically by just updating the state to the new vertex pointed by the edge selected by the agent through its action. The query and answer remains the same. Formally, the transition function is $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ defined by $\delta(S, A) = (v, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}}, \mathrm{e}_{2\mathrm{q}})$, where $S = (\mathrm{e}_{\mathrm{t}}, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}}, \mathrm{e}_{2\mathrm{q}})$ and $A = (\mathrm{e}_{\mathrm{t}}, r, v))$.

Rewards. We only have a terminal reward of +1 if the current location is the correct answer at the end and 0 otherwise. To elaborate, if $S_T = (\mathrm{e}_{\mathrm{t}}, \mathrm{e}_{1\mathrm{q}}, \mathrm{r}_{\mathrm{q}}, \mathrm{e}_{2\mathrm{q}})$ is the final state, then we receive a reward of +1 if $\mathrm{e}_{\mathrm{t}} = \mathrm{e}_{2\mathrm{q}}$ else 0.=, i.e. $R(S_T) = \mathbb{I}\{\mathrm{e}_{\mathrm{t}} = \mathrm{e}_{2\mathrm{q}}\}$.

2.2 Policy Network

To solve the finite horizon deterministic partially observable Markov decision process described above, we aim to design a randomized history-dependent policy $\pi = (\mathbf{d_1}, \mathbf{d_2}, ..., \mathbf{d_{T-1}})$, where $\mathbf{d_t} : H_t \to \mathcal{P}(\mathcal{A}_{S_t})$ and history $H_t = (H_{t-1}, A_{t-1}, O_t)$ is just the sequence of observations and actions taken. We restrict ourselves to the function class expressed by long short-term memory network (LSTM) (Hochreiter & Schmidhuber, 1997) for learning the randomized history-dependent policy.

An agent based on LSTM encodes the history H_t as a continuous vector $\mathbf{h_t} \in \mathbb{R}^{2d}$. We also have embedding matrix $\mathbf{r} \in \mathbb{R}^{|\mathcal{R}| \times d}$ and $\mathbf{e} \in \mathbb{R}^{|\mathcal{E}| \times d}$ for the binary relations and entities respectively. The history embedding for $H_t = (H_{t-1}, A_{t-1}, O_t)$ is updated according to LSTM dynamics:

$$\mathbf{h_t} = \text{LSTM}\left(\mathbf{h_{t-1}}, [\mathbf{a_{t-1}}; \mathbf{o_t}]\right) \tag{1}$$

where $\mathbf{a_{t-1}} \in \mathbb{R}^d$ and $\mathbf{o_t} \in \mathbb{R}^d$ denote the vector representation for action/relation at time $t-1$ and observation/entity at time t respectively and $[;]$ denote vector concatenation. To elucidate, $\mathbf{a_{t-1}} = \mathbf{r}_{A_{t-1}}$, i.e. the embedding of the relation corresponding to label of the edge the agent chose at time $t-1$ and $\mathbf{o_t} = \mathbf{e}_{\mathrm{e}_t}$ if $O_t = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q})$ i.e. the embedding of the entity corresponding to vertex the agent is at time t.

Based on the history embedding $\mathbf{h_t}$, the policy network makes the decision to choose an action from all available actions $(\mathcal{A}_{S_t})$ *conditioned* on the query relation. Recall that each possible action represents an outgoing edge with information of the edge relation label l and destination vertex/entity d. So embedding for each $A \in \mathcal{A}_{S_t}$ is $[\mathbf{r_l}; \mathbf{e_d}]$, and stacking embeddings for all the outgoing edges we obtain the matrix $\mathbf{A_t}$. The network taking these as inputs is parameterized as a two-layer feed-forward network with ReLU nonlinearity which takes in the current history representation $\mathbf{h_t}$ and the embedding for the query relation $\mathbf{r_q}$ and outputs a probability distribution over the possible actions from which a discrete action is sampled. In other words,

$$\begin{aligned}
&\mathbf{d_t} = \text{softmax}\left(\mathbf{A_t}(\mathbf{W_2}\text{ReLU}\left(\mathbf{W_1}\left[\mathbf{h_t}; \mathbf{o_t}; \mathbf{r_q}\right]\right))\right) \\
&A_t \sim \text{Categorical}\left(\mathbf{d_t}\right)
\end{aligned}$$

Note that the nodes in $\mathcal{G}$ do not have a fixed ordering or number of edges coming out from them. The size of matrix $\mathbf{A_t}$ is $|\mathcal{A}_{S_t}| \times 2d$, so the decision probabilities d_t lies on simplex of size $|\mathcal{A}_{S_t}|$. Also the procedure above is invariant to order in which edges are presented as desired and falls in purview of neural networks designed to be permutation invariant Zaheer et al. (2017). Finally, to summarise, the parameters of the LSTM, the weights $\mathbf{W_1}$, $\mathbf{W_2}$, the corresponding biases (not shown above for brevity), and the embedding matrices form the parameters θ of the policy network.

2.3 TRAINING

For the policy network (π_θ) described above, we want to find parameters θ that maximizes the expected reward:

$$J(\theta) = \mathbb{E}_{(e_1, r, e_2) \sim D} \mathbb{E}_{A_1, .., A_{T-1} \sim \pi_\theta}[R(S_T) | S_1 = (e_1, e_1, r, e_2)]$$

where we assume there is a true underlying distribution $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2) \sim D$. To solve this optimization problem, we employ REINFORCE (Williams, 1992) as follows:

- The first expectation is replaced with empirical average over the training dataset.
- For the second expectation, we approximate by running multiple rollouts for each training example. The number of rollouts is fixed and for all our experiments we set this number to 20.
- For variance reduction, a common strategy is to use an additive control variate baseline (Hammersley, 2013; Fishman, 2013; Evans & Swartz, 2000). We use a moving average of the cumulative discounted reward as the baseline. We tune the weight of this moving average as a hyperparameter. Note that in our experiments we found that learnt baseline performed similarly, but we finally settled for cumulative discounted reward as the baseline owing to its simplicity.
- To encourage the policy to sample more diverse paths rather than sticking with a few, we add an entropy regularization term to our cost function after multiplying it by a constant (β). We treat β as a hyperparameter to control the exploration exploitation trade-off.

Experimental Details We choose the relation and embedding dimension size as 200. The action embedding is formed by concatenating the entity and relation embedding. We use a 3 layer LSTM with dimension size of 400. The hidden layer size of MLP (weights $\mathbf{W_1}$ and $\mathbf{W_2}$) is set to 400. We use Adam (Kingma & Ba, 2014) with the default parameters in REINFORCE for the update. The best hyperparameter values can be found in appendix.

Dataset	#entities	#relations	#facts	#queries	#degree	
					avg.	median
COUNTRIES	272	2	1158	24	4.35	4.0
UMLS	135	49	5,216	661	38.63	28
KINSHIP	104	26	10686	1074	82.15	82
WN18RR	40,945	15	86,835	3134	2.19	2
NELL-995	75,492	200	154,213	3992	4.07	1
FB15K-237	14,505	237	272,115	20,466	19.74	14
WikiMovies	43,230	9	196,453	9952	6.65	4

Table 1: Statistics of various datasets used in experiments.

Corpus		Metric	Model				
			NeuralLP	**ComplEx**	**NTP**	**NTP-λ**	**MINERVA**
COUNTRIES	S1	AUC-PR	-	99.37±0.4	90.83±15.4	**100.0±0.0**	**100.0±0.0**
	S2		-	87.95±2.8	87.4±11.7	**93.04±0.4**	91±0.01
	S3		-	48.44±6.3	56.68±17.6	77.26±17.0	**93±0.01**
UMLS		HITS@1	-	0.47	0.47	0.51	**0.53**
		HITS@3	-	0.63	0.60	0.64	**0.67**
		HITS@10	0.70	0.80	0.79	0.81	0.81
		MRR	-	0.58	0.57	0.60	**0.64**
		MR	-	-	-	-	1.0
KINSHIP		HITS@1	-	0.34	0.24	0.39	**0.46**
		HITS@3	-	0.49	0.40	0.47	**0.62**
		HITS@10	0.73	0.74	0.60	0.71	**0.76**
		MRR	-	0.46	0.36	0.48	**0.55**
		MR	-	-	-	-	2.0

Table 2: Performance on COUNTRIES, UMLS and KINSHIP datasets. Our model consistently outperforms baselines across datasets and can learn complex logical rules as seen e.g. in results of COUNTRIES(S3). NEURAL LP only report HITS@10 scores on two datasets.

3 DATA

We test our model on the following query answering datasets. (a) COUNTRIES (Bouchard et al., 2015), (b) Alyawarra kinship (KINSHIP), (c) Unified Medical Language Systems (UMLS) (Kok & Domingos, 2007) (d) WN18RR (Dettmers et al., 2017), (e) NELL-995, (f) FB15k-237 (g) WikiMovies (Miller et al., 2016). We also test on a synthetic grid world dataset released by Yang et al. (2017) to test the ability of the model to learn rules of long length.

The COUNTRIES dataset is carefully designed to explicitly test the logical rule learning and reasoning capabilities of link prediction models. The dataset has 3 tasks (S1-3 in table 2) each requiring reasoning steps of increasing length and difficulty (see Rocktäschel & Riedel (2017) for more details about the tasks). We also test our model on existing large and challenging KG datasets ((d) - (f)). WN18RR is created from the original WORDNET18 dataset by removing test triples which can be answered trivially, making the datasets more realistic and challenging. Additionally, we test our model on a question answering dataset - WikiMovies (Miller et al., 2016) where the query is in *natural language* but the answers can be found in an accompanying KB. Table 1 report the various statistics of the datasets.

4 EXPERIMENTS

4.1 KNOWLEDGE GRAPH QUERY ANSWERING

This section describes the experimental results on the various knowledge graph query answering datasets. During inference, we do beam search with a beam width of 40 and rank entities by the probability of the trajectory the model took to reach the entity. Our evaluation metrics are various widely used ranking metrics for KB completion such as HITS@k, mean reciprocal rank (MRR), median rank (MR) and mean average precision (MAP). For grid world and Wikimovies, we report accuracy (or HITS@1).

Task	DeepPath	MINERVA
athleteplaysinleague	0.960	0.970
worksfor	0.711	**0.825**
organizationhiredperson	0.742	**0.851**
athleteplayssport	0.957	0.985
teamplayssport	0.738	**0.846**
personborninlocation	0.795	0.793
athletehomestadium	0.890	0.895
organizationheadquarteredincity	0.790	**0.946**
athleteplaysforteam	0.750	**0.824**

Table 3: MAP scores for different query relations on the NELL-995 dataset. Note that in this comparison, MINERVA refers to only a single learnt model for all query relations which is competitive with individual DeepPath models trained separately for each query relation.

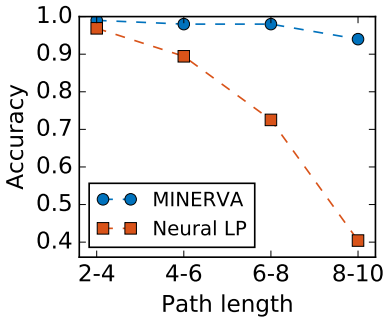

Figure 2: Grid world experiment: We significantly outperform NeuralLP for longer path lengths.

COUNTRIES, KINSHIP, UMLS. We first test MINERVA on the COUNTRIES dataset which is explicitly designed to test the ability of models to learn logical rules. It contains countries, regions and subregions as entities. The queries are of the form LocatedIn(c, ?) and the answer is a region. For example, LocatedIn(Egypt, ?) with the answer as Africa. Our experimental settings and scores are directly comparable to NTP and ComplEx (Trouillon et al., 2016). NTP-λ is a NTP model trained with an additional objective function of ComplEx. We also compare MINERVA against Neural LP (Yang et al., 2017) on the UMLS and KINSHIP datasets.

The evaluation metric we report is HITS@k - which is the percentage of correct entities ranked in top-k. For the COUNTRIES dataset, we report the area under the precision-recall curve for comparing with the baselines. Following the design of the COUNTRIES dataset, for task S1 and S2, we set the maximum path length $T = 2$ and for S3, we set $T = 3$.

Table 2 shows that MINERVA outperforms all the baseline models except on the task S2 of COUNTRIES, where the ensemble model NTP-λ outperforms it, albeit with a higher variance across runs. Our gains are much more prominent in task S3, which is the hardest among all the tasks. We similarly outperform NeuralLP on the UMLS and KINSHIP datasets.

NELL-995 We also compare MINERVA to DeepPath. For a fair comparison, we only rank the answer entities against the negative examples in the dataset used in their experiments[1] and report the mean average precision (MAP) scores for each query relation. DeepPath feeds the paths its agent gathers as input features to the path ranking algorithm (PRA) (Lao et al., 2011), which trains a per-relation classifier. But unlike them, we train one model which learns for all query relations. If our agent is not able to reach the correct entity or one of the negative entities, the corresponding query gets a score of negative infinity. As show in table 5, we outperform them or achieve comparable performance for all the query relations For this experiment, we set the maximum length $T = 3$.

Query Answering on NELL-995: Since the task of query answering is more challenging than fact prediction, we also test MINERVA for query answering on NELL-995 where the target entity is unknown. Since there are no published baselines on this task, we compare to a highly tuned implementation of DistMult (which achieves very high performance on other datasets). DistMult achieves a HITS@10 performance of 79.504. In comparison MINERVA achieves a score of 80.99. Moreover, MINERVA is more efficient at inference time since it does not have to compare against all the 75K entities in the dataset. The wall-clock inference time on the test set of NELL-995 for a GPU implementation of DistMult is 115s whereas that of MINERVA is 35s.

WN18RR Next we test MINERVA on another large KB dataset – WN18RR. On this dataset, we compare with three recently proposed latent factorization model – (a) ConvE (Dettmers et al., 2017), (b) DistMult (Yang et al., 2015), (c) ComplEx (Trouillon et al., 2016). We report HITS at various k and we compare favorably with the state-of-the-art results of ComplEx in all settings (table 4). For this experiment, we set the maximum length $T = 3$.

Model	HITS@1	HITS@3	HITS@10	MRR	MR
ConvE	0.306	0.360	0.411	0.342	-
DistMult	0.389	0.439	0.491	0.425	-
ComplEx	0.411	0.458	0.507	0.444	-
MINERVA	0.413	0.456	**0.513**	**0.448**	10.0

Table 4: Performance on WN18RR

[1] We are grateful to Xiong et al. (2017) for releasing the negative examples used in their experiments.

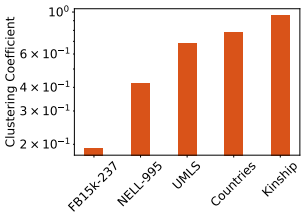

Figure 3: Network avg. cluster coefficient of various datasets

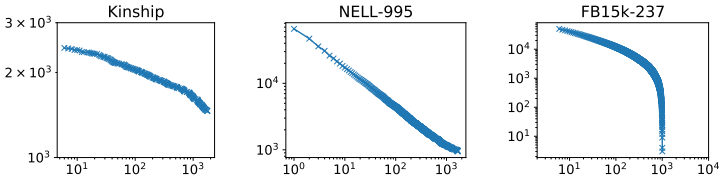

Figure 4: Count of number of unique path types of length 3 which occur more than 'x' times in various datasets. In Kinship and NELL-995, there are more than 10^3 path types which occur more than 10^3 times, however for FB15k-237, we see a sharp decrease as 'x' becomes higher, suggesting that path types do not repeat often.

FB15k-237 We test MINERVA on yet another popular KB dataset FB15K-237. The baselines are the same as before, however our implementation of DistMult gave a score of 56.8 HITS@10 which, to our knowledge, is the highest score reported on this dataset[2].The performance of ConvE and ComplEx are taken from Dettmers et al. (2017). Even though MINERVA performs comparably to ConvE and ComplEx, the results are significantly behind the performance of DistMult.

Model	HITS@10	MRR	MR
ConvE	0.458	0.301	-
DistMult	**0.568**	0.370	**8.0**
ComplEx	0.419	0.240	-
MINERVA	0.456	0.293	18.0

Table 5: Performance on FB15K-237

Upon delving more into the structure of knowledge graph derived from FB15K-237, we found few interesting characteristics of the dataset. As a prelude, we would like to describe a long existing concept in graph theory – clustering coefficient (Holland & Leinhardt, 1971; Watts & Strogatz, 1998). Clustering coefficient (τ) of a graph measures whether groups of nodes form 'tightly knit' communities - i.e. whether groups of nodes tend to cluster together. A high τ implies the presence of higher number of densely connected groups of nodes. For instance, if we consider three nodes A, B and C, a high τ means with high probability whenever three nodes are connected as A — B — C, it implies nodes A — C are also connected forming a triangle. Intuitively, MINERVA can use such closed shapes to learn paths such as (A — B — C) to predict the answer of the query, i.e. the third node (C). The clustering coefficient also extends from triangles to cliques of arbitrary size (Watts & Strogatz, 1998). Figure 3 plots τ for various datasets. We find that FB15k-237 has the least clustering coefficient (0.19) among all datasets. This means that the dataset has sparse neighborhoods and hence MINERVA finds it difficult to learn logical rules. We also check the frequency of occurrence of various unique paths (types). We define a path type as the sequence of relations (ignoring the entities) in a path. Intuitively, a predictive path which generalizes across queries will occur many number of times in the graph. Figure 4 shows the plot. As we can see, the characteristics of FB15k-237 is quite different from other datasets. Path types do not repeat often, making it hard for MINERVA to learn paths that generalizes. We also do further analysis of the types of various query relations in FB15k-237 below.

Analysis of query relations of FB15k-237: Next we analyzed the type of query relations on the FB15k-237 dataset. Following Bordes et al. (2013), we categorized the query relations into (M)any to 1, 1 to M or 1 to 1 relations. An example of a M to 1 relation would be '/people/profession' (What is the profession of person 'X'?). An example of 1 to M relation would be /music/instrument/instrumentalists ('Who plays the music instrument X?') or '/people/ethnicity/people' ('Who are people with ethnicity X?'). From a query answering point of view, the answer to these questions is a list of entities. However, during evaluation time, the model is evaluated based on whether it is able to predict the one target entity which is in the query triple. Also, since MINERVA outputs the end points of the paths as target entities, it is sometimes possible that the particular target entity of the triple does not have a path from the source entity (however there are paths to other 'correct' answer entities). Table 8 (in appendix) shows few other examples of relations belonging to different classes.

Following Bordes et al. (2013), we classify a relation as 1-to-M if the ratio of cardinality of tail to head entities is greater than 1.5 and as M-to-1 if it is lesser than 0.67. In the validation set of FB15k-237, 54% of the queries are 1-to-M, whereas only 26% are M-to-1. Contrasting it with NELL-995, 27% are 1-to-M and 36% are M-to-1 or UMLS or KINSHIP where only 18% and 32% of the relations are 1-to-M. Table 9 (in appendix) shows few relations from FB15k-237 dataset which have high tail-to-head ratio. The average ratio for 1-TO-M relations in FB15k-237 is **13.39** (substantially higher than 1.5).

[2]We are aware of the high variance of DistMult scores reported on FB15k-237 by several papers, but to ensure fairness we report the high scores our in-house implementation achieved.

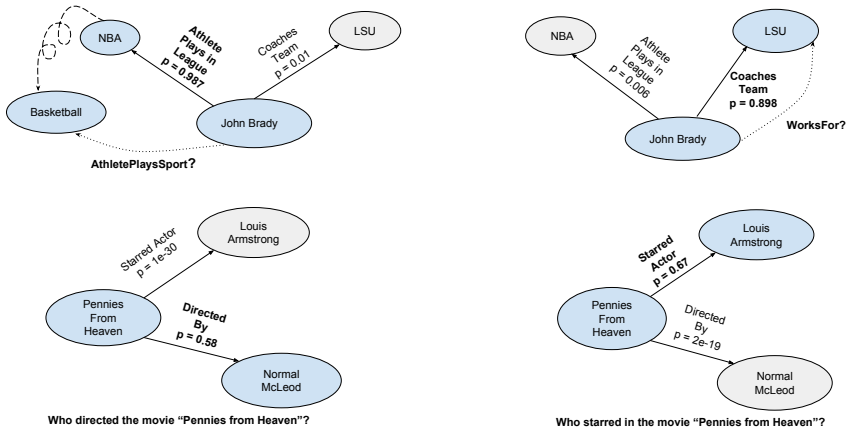

Figure 5: Based on the query relation our agent assigns different probabilities to different actions. The dashed edges in the top row denote query relation. Examples in the bottom row are from the WikiMovies dataset and hence the questions are partially structured.

4.2 Grid World Path Finding

As we empirically find and also noted by previous work (Rocktäschel & Riedel, 2017; Das et al., 2017; Yang et al., 2017), often the reasoning chains required to answer queries in KB is not too long (restricted to 3 or 4 hops). To test if our model can learn long reasoning paths, we test our model on a synthetic 16-by-16 grid world dataset created by Yang et al. (2017), where the task is to navigate to a particular cell (answer entity) starting from a random cell (start entity) by following a set of directions (query relation). The KB consists of atomic triples of the form ((2,1), North, (1,1)) – entity (1,1) is north of entity (2,1). The queries consists of a sequence of directions (e.g. North, SouthWest, East). The queries are classified into classes based on the path lengths. Figure 2 shows the accuracy on varying path lengths. Compared to Neural LP, MINERVA is much more robust for queries which require longer path lengths showing a very little degrade in performance for even the longest path length in the dataset.

4.3 Partially Structured Queries

Queries in KB datasets are structured in the form of triples. However, this is unsatisfactory since for most real applications, the queries appear in natural language. As a first step in this direction, we extend MINERVA to take in "partially structured" queries. We use the WikiMovies dataset (Miller et al., 2016) which contains questions in natural language albeit generated by templates created by human annotators. An example question from the dataset is "Which is a film written by Herb Freed?". WikiMovies also has an accompanying KB which can be used to answer all the questions.

Model	Accuracy
Memory Network	78.5
QA system	93.5
Key-Value Memory Network	93.9
Neural LP	94.6
MINERVA	**96.7**

Table 6: Performance on WikiMovies

We link the entity occurring in the question to the KB via simple string matching. To form the vector representation of the query relation, we design a simple question encoder which computes the average of the embeddings of the question words. The word embeddings are learned from scratch and we do not use any pretrained embeddings. We compare our results with those reported in Yang et al. (2017) (table 6). We got the best result using $T = 1$, suggesting that WikiMovies is not the best testbed for multihop reasoning, but this experiment is a promising first step towards the realistic setup of having textual queries and knowledge bases.

5 Analysis

Effectiveness of Remembering Path History. MINERVA encodes the history of decisions it has taken in the past using LSTMs. To test the importance of remembering the sequence of decisions,

(i) **Can learn general rules:**

(S1) LocatedIn(X, Y) ← LocatedIn(X, Z) & LocatedIn(Z, Y)
(S2) LocatedIn(X, Y) ← NeighborOf(X, Z) & LocatedIn(Z, Y)
(S3) LocatedIn(X, Y) ← NeighborOf(X, Z) & NeighborOf(Z, W) & LocatedIn(W, Y)

(ii) **Can learn shorter path:** Richard F. Velky $\xrightarrow{\text{WorksFor}}$?

Richard F. Velky $\xrightarrow{\text{PersonLeadsOrg}}$ Schaghticokes $\xrightarrow{\text{NO-OP}}$ Schaghticokes $\xrightarrow{\text{NO-OP}}$ Schaghticokes

(iii) **Can recover from mistakes:** Donald Graham $\xrightarrow{\text{WorksFor}}$?

Donald Graham $\xrightarrow{\text{OrgTerminatedPerson}}$ TNT Post $\xrightarrow{\text{OrgTerminatedPerson}^{-1}}$ Donald Graham $\xrightarrow{\text{OrgHiredPerson}}$ Wash Post

Table 7: A few example of paths found by MINERVA on the COUNTRIES and NELL. MINERVA can learn general rules as required by the COUNTRIES dataset (example (i)). It can learn shorter paths if necessary (example (ii)) and has the ability to correct a previously taken decision (example (iii))
.

we did an ablation study in which the agent chose the next action based on only local information i.e. current entity and query and did not have access to the history h_t. For the KINSHIP dataset, we observe a 27% points decrease in HITS@1 (0.184 v/s 0.46) and 13% decrease in HITS@10 (0.63 v/s 0.76). For grid-world, it is also not surprising that we see a big drop in performance. The final accuracy is 0.23 for path lengths 2-4 and 0.04 for lengths 8-10. For NELL, the performance dropped from 0.576 to 0.564 and for FB15k-237 the HITS@10 performance dropped from 0.456 to 0.408.

NO-OP and Inverse Relations. At each step, MINERVA can choose to take a NO-OP edge and remain at the same node. This gives the agent the flexibility of taking paths of variable lengths. Some questions are easier to answer than others and require lesser steps of reasoning and if the agent reaches the answer early, it can choose to remain there. Example (i) in table 7 shows such an example. Similarly inverse relation gives the agent the ability to recover from a potentially wrong decision it has taken before. Example (ii) shows such an example, where the agent took a incorrect decision at the first step but was able to revert the decision because of the presence of inverted edges.

Query based Decision Making. At each step before making a decision, our agent conditions on the query relation. Figure 5 shows examples, where based on the query relation, the probabilities are peaked on different actions. For example, when the query relation is WorksFor, MINERVA assigns a much higher probability of taking the edge CoachesTeam than AthletePlaysInLeague. We also see similar behavior on the WikiMovies dataset where the query consists of words instead of fixed schema relation.

Inference Time. MINERVA is efficient at inference time since it has to essentially search for answer entities in its local neighborhood, whereas previous methods rank all the entities in the dataset. For instance, on the test dataset of WN18RR, the wall clock running time of MINERVA is 63 seconds whereas that of a GPU implementation of DistMult is 211 seconds (with the maximum batch size).

6 RELATED WORK

Learning vector representations of entities and relations using tensor factorization (Nickel et al., 2011; 2012; Bordes et al., 2013; Riedel et al., 2013; Nickel et al., 2014; Yang et al., 2015) or neural methods (Socher et al., 2013; Toutanova et al., 2015; Verga et al., 2016) has been a popular approach to reasoning with a knowledge base. However, these methods cannot capture more complex reasoning patterns such as those found by following inference paths in KBs. Multi-hop link prediction approaches (Lao et al., 2011; Neelakantan et al., 2015; Guu et al., 2015; Toutanova et al., 2016; Das et al., 2017) address the problems above, but the reasoning paths that they operate on are gathered by performing random walks independent of the type of query relation. Lao et al. (2011) further filters

paths from the set of sampled paths based on the restriction that the path must end at one of the target entities in the training set and are within a maximum length. These constraints make them query dependent but they are heuristic in nature. Our approach eliminates any necessity to pre-compute paths and learns to efficiently search the graph conditioned on the input query relation.

Inductive Logic Programming (ILP) (Muggleton et al., 1992) aims to learn general purpose predicate rules from examples and background knowledge. Early work in ILP such as FOIL (Quinlan, 1990), PROGOL (Muggleton, 1995) are either rule-based or require negative examples which is often hard to find in KBs (by design, KBs store true facts). Statistical relational learning methods (Getoor & Taskar, 2007; Kok & Domingos, 2007; Schoenmackers et al., 2010) along with probabilistic logic (Richardson & Domingos, 2006; Broecheler et al., 2010; Wang et al., 2013) combine machine learning and logic but these approaches operate on symbols rather than vectors and hence do not enjoy the generalization properties of embedding based approaches.

Neural Theorem Provers (NTP) (Rocktäschel & Riedel, 2017) and Neural LP (Yang et al., 2017) are two recent methods in learning logical rules that can be trained end-to-end with gradient based learning. NTPs are constructed by Prolog's backward chaining inference method. It operates on vectors rather than symbols, thereby providing a success score for each proof path. However, since a score can be computed between any two vectors, the computation graph becomes quite large because of such *soft-matching* during substitution step of backward chaining. For tractability, it resides to heuristics such as only keeping the top-K scoring proof paths, but it loses any guarantee of computing exact gradients. Also the efficacy of NTPs has yet to be shown on large KBs. Neural LP introduces a differential rule learning system using operators defined in TensorLog (Cohen, 2016) and has a LSTM based controller and a differentiable memory component (Graves et al., 2014; Sukhbaatar et al., 2015) and the rule scores are calculated via attention. Even though, differentiable memory allows the network to be trained end to end, it necessitates accessing the entire memory which can be computationally expensive. RL approaches which can make hard selection of memory (Zaremba & Sutskever, 2015) are computationally attractive. MINERVA uses a similar hard selection of relation edges to walk on the graph. More importantly, MINERVA outperforms both these methods on their respective benchmark datasets.

DeepPath (Xiong et al., 2017) uses RL based approaches to find paths in KBs. However, the state of their MDP requires the target entity to be known in advance and hence their path finding strategy is dependent on knowing the answer entity. MINERVA does not need any knowledge of the target entity and instead learns to find the answer entity among all entities. DeepPath, additionally feeds its gathered paths to Path Ranking Algorithm (Lao et al., 2011), whereas MINERVA is a complete system trained to do query answering. DeepPath also uses fixed pretrained embeddings for its entity and relations. Lastly, on comparing MINERVA with DeepPath in their experimental setting on the NELL dataset, we match their performance or outperform them. MINERVA is also similar to methods for learning to search for structured prediction (Collins & Roark, 2004; Daumé III & Marcu, 2005; Daumé III et al., 2009; Ross et al., 2011; Chang et al., 2015). These methods are based on imitating a reference policy (oracle) which make near-optimal decision at every step. In our problem setting, it is unclear what a good reference policy would be. For example, a shortest path oracle between two entities would be bad, since the answer providing path should depend on the query relation.

7 CONCLUSION

We explored a new way of automated reasoning on large knowledge bases in which we use the knowledge graphs representation of the knowledge base and train an agent to walk to the answer node conditioned on the input query. We achieve state-of-the-art results on multiple benchmark knowledge base completion tasks and we also show that our model is robust and can learn long chains-of-reasoning. Moreover it needs no pretraining or initial supervision. Future research directions include applying more sophisticated RL techniques and working directly on textual queries and documents.

REFERENCES

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *ICDM*, 2008.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

Guillaume Bouchard, Sameer Singh, and Theo Trouillon. On approximate reasoning capabilities of low-rank vector spaces. *AAAI Spring Symposium*, 2015.

Matthias Broecheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *UAI*, 2010.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, Jr., and Tom M. Mitchell. Toward an Architecture for Never-ending Language Learning. In *AAAI*, 2010.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

William Cohen. Tensorlog: A differentiable deductive database. *arXiv:1605.06523*, 2016.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *ACL*, 2004.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *EACL*, 2017.

Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*, 2005.

Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 2009.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. *arXiv:1707.01476*, 2017.

Michael Evans and Timothy Swartz. *Approximating integrals via Monte Carlo and deterministic methods*. OUP Oxford, 2000.

George Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media, 2013.

Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. MIT press, 2007.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *EMNLP*, 2015.

John Hammersley. *Monte carlo methods*. Springer Science & Business Media, 2013.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Paul W Holland and Samuel Leinhardt. Transitivity in structural models of small groups. *Comparative Group Studies*, 1971.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

Stanley Kok and Pedro Domingos. Statistical predicate invention. In *ICML*, 2007.

Ni Lao, Tom Mitchell, and William Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, 2011.

John McCarthy. *Programs with common sense*. RLE and MIT Computation Center, 1960.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *EMNLP*, 2016.

Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *HLT-NAACL*, 2013.

Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 1995.

Stephen Muggleton, Ramon Otero, and Alireza Tamaddoni-Nezhad. *Inductive logic programming*. Springer, 1992.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. In *ACL*, 2015.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, 2011.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *WWW*, 2012.

Maximilian Nickel, Xueyan Jiang, and Volker Tresp. Reducing the rank in relational factorization models by including observable patterns. In *NIPS*, 2014.

Nils J Nilsson. Logic and artificial intelligence. *Artificial intelligence*, 1991.

J Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 1990.

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 2006.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation extraction with matrix factorization and universal schemas. In *NAACL*, 2013.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NIPS*, 2017.

Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Stefan Schoenmackers, Oren Etzioni, Daniel Weld, and Jesse Davis. Learning first-order horn clauses from web text. In *EMNLP*, 2010.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *KDD*, 2017.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, 2013.

Fabian Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *WWW*, 2007.

Sainbayar Sukhbaatar, Jason Weston, and Rob Fergus. End-to-end memory networks. In *NIPS*, 2015.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *EMNLP*, 2015.

Kristina Toutanova, Victoria Lin, Wen-tau Yih, Hoifung Poon, and Chris Quirk. Compositional learning of embeddings for relation paths in knowledge base and text. In *ACL*, 2016.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

Patrick Verga, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum. Multilingual relation extraction using compositional universal schema. In *NAACL*, 2016.

William Yang Wang, Kathryn Mazaitis, and William W Cohen. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *CIKM*, 2013.

Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *nature*, 1998.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 2017.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.

Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NIPS*, 2017.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. In *NIPS*, 2017.

Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *arXiv:1505.00521*, 2015.

(i) **M to 1**

Los Angeles Rams $\xrightarrow{\text{team plays sport}}$ American Football
The Walking Dead $\xrightarrow{\text{country of origin}}$ USA

(ii) **1 to M**

CEO $\xrightarrow{\text{job position in organization}}$ Merck & Co.
Traffic collision $\xrightarrow{\text{cause of death}}$ Albert Camus
Harmonica $\xrightarrow{\text{instrument played by musician}}$ Greg Graffin

Table 8: Few example facts belonging to m to 1, 1 to m relations in FB15k-237
.

Relation	tail/head
/people/marriage_union_type/unions_of_this_type./people/marriage/location_of_ceremony	129.75
/organization/role/leaders./organization/leadership/organization	65.15
/location/country/second_level_divisions	49.18
/user/ktrueman/default_domain/international_organization/member_states	36.5
/base/marchmadness/ncaa_basketball_tournament/seeds./base/marchmadness/ncaa_tournament_seed/team	33.6

Table 9: Few example 1-to-M relations from FB15k-237 with high cardinality ratio of tail to head.

8 APPENDIX

8.1 HYPERPARAMETERS

In our experiments, we tune our model over two hyper parameters, *viz.,* β which is the entropy regularization constant and λ which is the moving average constant for the REINFORCE baseline. The table 10 lists the best hyper parameters for all the datasets.

Dataset	β	λ	Path Length
UMLS	0.05	0	2
KINSHIP	0.05	0.05	2
Countries S1	0.01	0.1	2
Countries S2	0.02	0.02	2
Countries S3	0.01	0.01	3
WN18RR	0.05	0.05	3
NELL-995	0.05	0.02	3
FB15K-237	0.02	0.05	3
WIKIMOVIES	0.15	0	1

Table 10: Best hyper parameters