CS221

Evaluating RL-Based Approaches for Knowledge Graph Reasoning

Parthasarathy Suryanarayanan
Department of Computer Science
Stanford University
psuryan@stanford.edu

Hermann Qiu
Department of Computer Science
Stanford University
hq2128@stanford.edu

David Uvalle
Department of Computer Science
Stanford University
davidu@stanford.edu

https://github.com/psuryan/cs221-project

Abstract

Reasoning over knowledge graph [KG] is an active research topic. In this work, we investigate some of the reinforcement learning techniques for reasoning over KGs by studying the system presented in the paper "DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning"[1], and further investigate several hypotheses to improve the performance.

1. Introduction

KGs, as a type of knowledge representation, have gained much attention in natural language processing in recent years. KGs can effectively organize and represent knowledge so that they can be efficiently utilized in advanced applications such as question answering and recommender systems. In other words, Knowledge Graph brings the ability to represent entities and relations with high reliability, explainability, and reusability. Per Wikipedia, Google's Knowledge Graph "uses a graph database to provide structured and detailed information about the topic in addition to a list of links to other sites." This knowledge graph, built on top of a graph database, has allowed Google to focus its search on things – or concepts – and understand exactly what you're looking for based on context.

Reasoning over KGs is an active research topic, since it can obtain new knowledge and conclusions from existing data. Another purpose for reasoning is because KGs suffer from incomplete coverage [3] and knowledge in KGs needs to be expanded to cover more facts and reasoning is one way to do so. The ability to reason over learned knowledge is an innate ability for humans however it is a challenging task for machines for many reasons. The typical methods for reasoning over the knowledge graphs are either rule-based, distributed representation (embedding) based or based on deep learning. The paper summarized below approaches the reasoning problem from a reinforcement learning perspective which is both novel and intriguing.

The organization of the paper is as follows. In Section 2 we briefly show the landscape of related work. Section 3 gives a comprehensive overview of the DeepPath architecture and our approach for implementation. Section 4 describes the different experiments we conducted. Section 5 discusses our

observations. In Section 6, we summarize our main conclusions from this work and describe our vision for the future.

2. Background And Related Work

The Path-Ranking Algorithm (PRA) method [2] is a primary path-finding approach that uses random walks with restart strategies for multi-hop reasoning and there have been several improvements on top of this algorithm. More recently neural-based approaches, such as using CNN[7] and KG-embeddings[5] have been developed. Other approaches such as [15] train a separate recurrent model for each relational path. The method presented in [1] uses a single model for a given task that combines several reasoning paths.

3. Approach

The first of our two step approach is essentially studying the architecture carefully as specified in the paper. The second step is to perform an error analysis on the development set, understand the strengths and weaknesses of the model and come up with a few ideas that might help the performance.

3.1 Representation

DeepPath proposes a method for controllable multi-hop reasoning. The problem is framed to use reinforcement learning for path learning between entities. The KG is modeled as a partially observed Markov decision process (MDP), which is a 4-tuple: $< S, A, P, R >$ where S is the continuous state space, $A = \{a_1, a_2, ..., a_n\}$ is the set of all available actions, $P(S_{s+1} = s' \mid S_t = s, A_t = a)$ is the transition probability matrix, and $R(s, a)$ is the reward function for every (s, a) pairs.

The agent uses a handcrafted reward function that allows it to learn correct paths and its trained using policy gradient. The model generated was evaluated in two tasks ***Link Prediction*** and ***Fact Prediction*** by using MAP(mean average precision). We restricted our experiments and error analysis to the ***Fact Prediction*** task alone due to time constraints. The model framework is described in greater detail in section 3.4.

3.2 Baseline

We use the results from the RL-based method discussed in the original work as our baseline, although we were unable to perfectly reproduce the results presented in the paper. The KG reasoning evaluation tasks (such as ***Link Prediction*** and ***Fact Prediction***) that based on either directly using path formulas or by performing arithmetic over the KG embeddings. The DeepPath system demonstrates that the RL-based method outperforms the Path-Ranking algorithm (PRA)[9] method and KG embedding based method, such as TransE, TransH, TransR and TransD.

3.3 Oracle

The KG reasoning evaluation tasks are conducted on two different datasets FB15K-237[7] and NELL-995[1]. The FB15K-237 dataset is derived from Freebase and hence the oracle in this case is the crowdsourced ground-truth. Similarly the oracle in case of of the NELL-995 dataset, released as part of the DeepPath work, is the semi-automated NELL system.

3.4 Reinforcement Learning Framework

The RL system consists of two parts (see Figure 1). The first part is the external environment modeled as a partially observed MDP, and the second is the RL agent represented as a Policy Network.

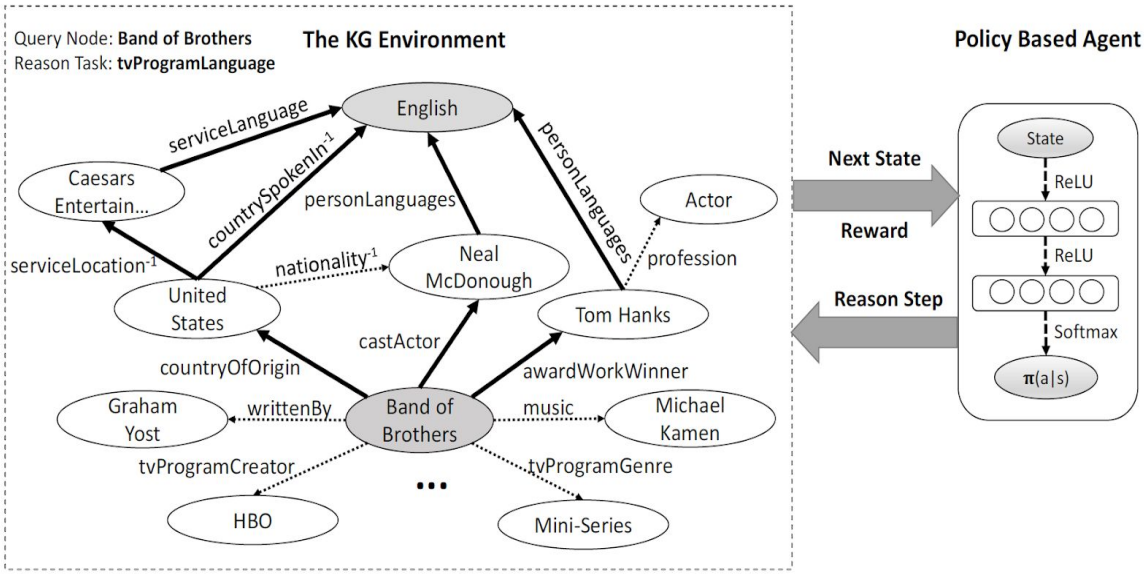

Figure 1: DeepPath system

Left: The KG environment modeled by MDP. The dotted arrows show the existing relation links in the KG and the bold arrows show the reasoning paths found by the RL agent.

States

In order to represent entities and their relationships, the framework uses translation-based embeddings similar to TransE[6] and TransH[12]. TransE/H model entities and their relationships such as given a tuple: $(head, label, tail)$, the embedding of the tail entity should be close to the head entity plus some vector that depends on the relationship label. In the framework, each state represents the agent position in the knowledge graph. After taking an action, the agent will move from one entity to another. The state vector at step t is given as follows:

$$s_t = (e_t,\ e_{target} - e_t)$$

Where e_t denotes the embeddings of the current entity and e_{target} denotes the embedding of the target entity. At the initial state $e_t = e_{source}$. It is to be noted that the reasoning relation is not encoded in the state and the RL agent discovers the reasoning semantics after the training with positive examples for a given relation.

Actions

Given the entity pairs $(e_s,\ e_t)$ with relation r, we want the agent to find the most informative (rewarding) paths linking these entity pairs. Beginning with the source entity e_s, the agent uses the policy network to pick the most promising relation to extend its path at each step until it reaches the target entity e_t. To keep the output dimension of the policy network consistent, the action space is defined as all the relations in the KG.

Rewards

The reward function is a composition of factors that contribute to improve the quality of the paths found by the RL agent. The reward factors are designed to encourage the agent to find predictive paths. Each factor of the reward function is described below:

1. Global accuracy

Given that the knowledge graph problem space is large, the agent can take more incorrect sequential decisions than correct ones. The first reward factor is defined as:

$$r_{\text{GLOBAL}} = \begin{cases} +1, & \text{if the path reaches } e_{target} \\ -1, & \text{otherwise} \end{cases}$$

2. Path efficiency

Short paths showed more reliable reasoning evidence than longer paths. Shorter chains also improve the efficiency of the reasoning by limiting the length of the agent interaction with the environment. The efficiency reward is defined as:

$$r_{efficiency} = \frac{1}{length(p)}$$

Where p is defined as a sequence of relations $r_1 \rightarrow r_2 \rightarrow r_3 \ldots \rightarrow r_n$

3. Path diversity

In order to encourage the agent to find diverse paths, we define a diversity reward function using the cosine similarity between the current path and the existing ones:

$$r_{diversity} = -\frac{1}{|F|} \sum_{i=1}^{|F|} cos(p, p_i)$$

Where $p = \sum_{i=1}^{n} r_i$ represents the path embedding for the relation chain $r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_n$.

3.5 Policy Network

A fully connected neural network is used to parameterize the policy function $\pi(s; \theta) = p(a|s; \theta)$ that maps the state vector s to a probability distribution over all possible actions. The neural network consists of two hidden layers, each followed by a rectifier nonlinearity layer (ReLU). The output layer is normalized using a softmax function.

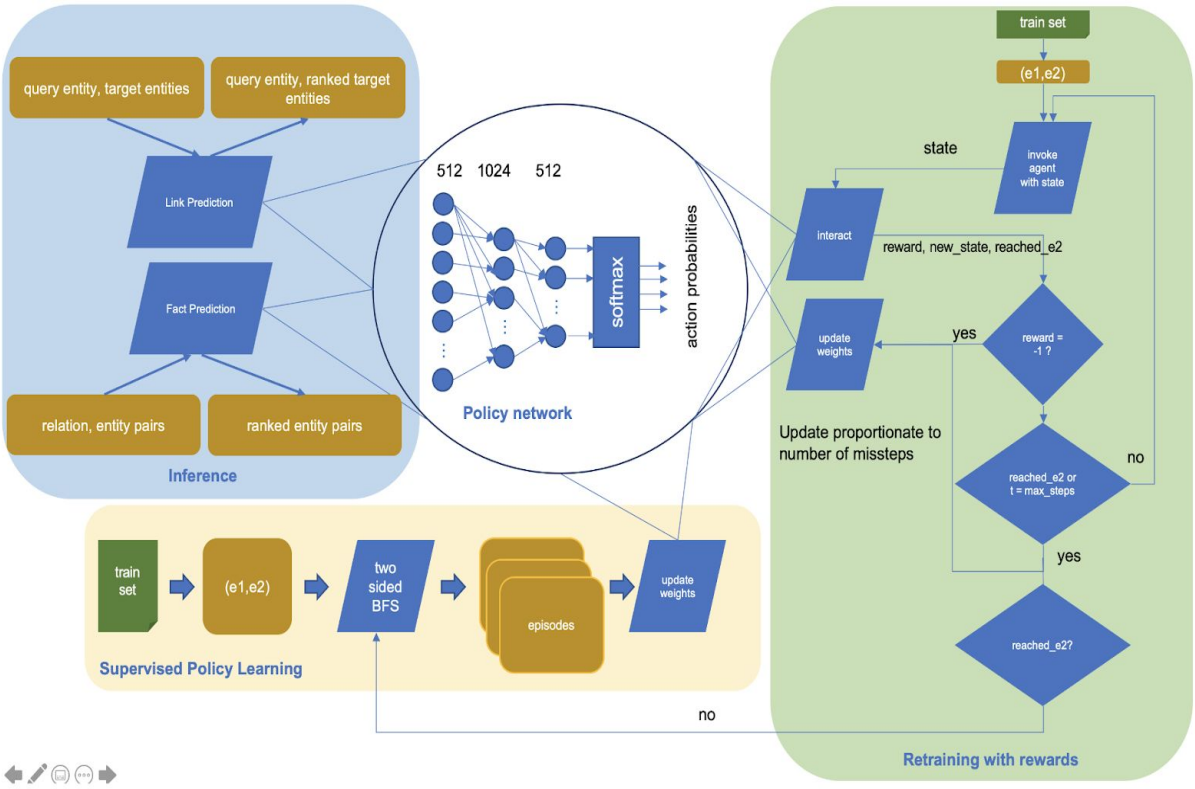

Figure 2: Model Training & Model Inference

3.6 Training

Supervised Policy Learning

In order to tackle the problem of large training space, a supervised policy is trained with randomized breadth-first search (BFS). For each training sample, a two-sided BFS is conducted to find same correct paths between the entities (e_{source} and e_{target}) using Monte-Carlo Policy Gradient [13]

$$J(\theta) = E_{a \sim \pi(a \mid s\,;\,\theta)}(\sum_{t} R_{st,\,at}) = \sum_{t} \sum_{a \in A} \pi(a \mid s_t;\,\theta)\, R_{st,\,a_t}$$

To mitigate the BFS's bias towards preferring shorter paths, a randomly picked intermediate node (e_{inter})is added to the path. Using the correct paths found by the randomized BFS, the policy network is updated with the approximate gradient

$$\nabla_\theta J(\theta) = \sum_{t} \sum_{a \in A} \pi(a \mid s_t;\,\theta) \nabla_\theta \log \pi(a \mid s_t\,;\,\theta) \approx \nabla_\theta \sum_{t} \log \pi(a = r_t \mid s_t\,;\,\theta)$$

This first round of training greatly improves the overall efficiency of the agent. After this step, the agent is retrained using a linear combination of reward functions (R_{total}) defined earlier.

Retraining with Rewards

For each relation, reasoning with one entity pair (e_{source} and e_{target}) is treated as an episode (bound to a $maxlength$). At each step, the agent picks the next relation using a stochastic policy $\pi(a \mid s)$, which is nothing but a probability distribution over all reasoning paths, to extend the current reasoning path and rewarded according to the eventual outcome. After each episode the policy network is updated using the gradient

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{t} \log \pi(a = r_t \mid s_t\,;\,\theta)\, R_{total}$$

3.7 Inference

During inference, given an entity pair, the reasoning paths learned by the RL agent are used as logical formulas to predict the relation link. For efficiency, each formula is verified using a bi-directional search to counter the asymmetric fan-out of relations. For example, verification using the inverse relation $personNationality^{-1}$, can greatly reduce huge number of neighbours obtained through $personNationality$ relation for the query entity "United States". Thus the number of intermediate nodes are significatively decreased.

4. Data and Experiments

4.1 Data

As stated earlier the KG reasoning evaluation tasks are conducted on two different datasets FB15K-237[7] and NELL-995[1]. The FB15K-237 dataset is derived from Freebase is a subset of a larger dataset FB15K [6]. The other dataset, NELL-995, that was created as part of the DeepPath paper by filtering the original NELL[11] dataset. Some of the processing done to the NELL-955 dataset includes only selecting triples with top-200 relations. Inverse triples were added some for each triple (h, r, t) there is a (t, r^{-1}, h) relation.

Dataset	# Entities	# Relations	# Triples
FB15K-237	14,505	237	310,116
NELL-995	75,492	200	154,213

Table 1: Datasets

We wish to note that all the analysis and experimentation was conducted on the NELL-995 dataset only, again due to time constraints. Some examples from this dataset are shown below.

Entity 1	Entity 2	Relation
concept:athlete:lebron	concept:stadiumoreventvenue:united_center	concept:athletehomestadium
concept:person:brooke	concept:city:york	concept:personborninlocation
concept:person:molly_moore	concept:automobilemaker:toyota_motor	concept:worksfor

Table 2: Examples from NELL dataset

4.2 Evaluation

The evaluation metric for both ***Link Prediction***, ***Fact Prediction*** is "mean average precision" (MAP) over the various relation tasks. Average Precision (AP) defined as following

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q}$$

where Q is the number of queries in the set and AveP(q) is the average precision (AP) for a given query, q. Mean average is a popular metric in information retrieval system where there is a notion of ranking.

4.3. Experiments

4.3.1 Efficiency improvements

The original implementation provided along the paper, had several issues - functional bugs and extremely poor training speeds due to inefficient implementation. We spent several days in fixing these bugs and implemented numerous speed improvements that has resulted in 15x speed increase in the training pipeline so far. These include

1. Identification and mitigation of severe I/O contention issues. As an example, for instance, we found that the entire train dataset was being re-loaded in every episode. The resulting I/O activity turned out to be a major bottleneck for training speeds. This resulted in a 10x improvement in speed.
2. Addition of caching results of the BFS:

   Step 1 (Supervised Policy Learning): The first step of the training pipeline includes the creation of a supervised model, in order to create that model a bi-directional BFS is run between the two entities to find if they can be connected through relations or not.

   Step 2 (Retraining with Rewards): In the second step of the pipeline a RL agent learns to find good paths between entities and if it fails, a teaching procedure updates the model with a ground truth example using BFS. Since the BFS was computed in the first part of the pipeline, we implemented a cache to persist the paths found in step 1, so that the RL agent doesn't need to recompute them again. The improvement described resulted in an additional 5x reduction of training time.

4.3.2 Experimental setup

In this section, we describe the experiments conducted by us to evaluate the performance of the DeepPath model. The Figure 3 shows a typical experimental run for supervised training (**Step 1**, described above).

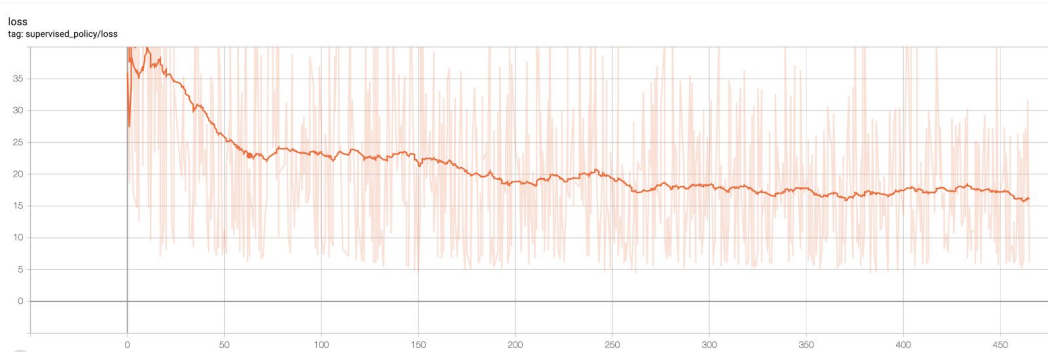

Figure 3: Supervised Policy Learning

The Figure 3 shows the loss as a function of training episodes for one of the reasoning tasks *athlete-home-stadium*. As expected, the supervised policy learning results agent learning reasonable priors for the reasoning paths but does not add a lot of value as shown by marginal decrease loss that quickly stabilizes.

The Figure 4 shows a typical run of the **Step 2** - retraining with reward. As one would expect, the losses are smaller in magnitude and drop steeply with more episodes. Similar to figure 3, the loss fluctuates significantly from episode to episode.

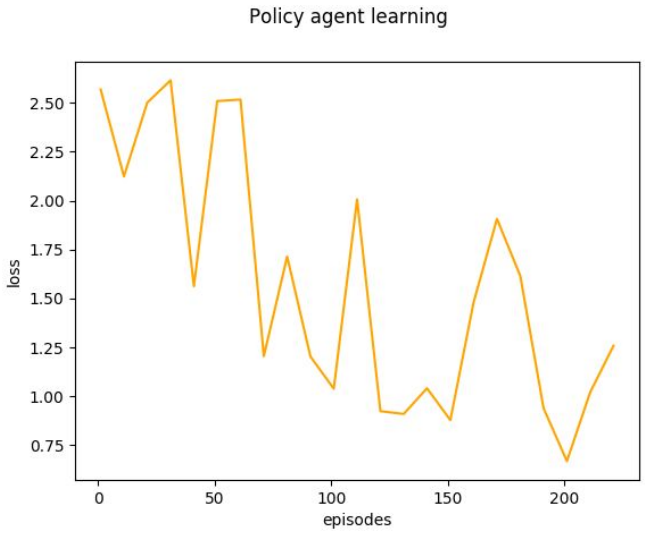

Figure 4: Retraining with Rewards

All the experiments were run with the hyper parameters specified in the paper. As subset of them are given below for reference.

state_dim - dimension of the state of the agent	200
action_space - dimension of the possible actions that could be taken at each state	400
max_steps - maximum number of steps for the episode to be terminated	50

Table 3: Hyperparameters

4.3.3 Supervised Policy Learning Experiments

We wanted to investigate improving the supervised policy learning. We conducted three separate experiments, each introducing some kind of data augmentation

1. **More intermediate nodes** - in the original implementation for every training pair, the BFS produced 5 different good paths corresponding to 5 different intermediate nodes. We increased the number of good paths from 5 to 20 to teach the agent different ways of reaching e_{target} given an e_{source}
2. **More training samples** - although a large number of training pairs were available for each task, the original implementation capped the number of training samples for each task to 500. We hypothesize that it is due to the poor training speeds in the original implementation. Given our efficiency improvements described in section 4.3.1, we removed this cap and let the model be trained over all the training data available for each task.
3. **Longer training** - We experimented with repeating all the episodes already seen by the model in each epoch for a several epochs.

4.3.3 Retraining with Rewards Experiment - Modified Reward Factor Function

Figure 5: Experiment with a Modified Reward Factor Function

Originally, one of the reward factors, called Path Efficiency, penalizes over the length of path by doing

$$r_{efficiency} = \frac{1}{length(p)}$$

While the short paths show more reliable reasoning evidence than longer paths, we wanted to investigate the effect of adjusted the penalization in either direction on the performance. We tested the following two smooth functions on one single task. Please note that we enforce r(1)=1 for all three factors and also limit the factor value to be within (0,1] assuming length can not be zero if it's a successful path. $r_{efficiency+}$ penalizes much more than the original factor, while $r_{efficiency-}$ the opposite.

$$r_{efficiency+} = 2 - \frac{2}{1 + exp(-length(p) + 1)}$$

$$r_{efficiency-} = \frac{1}{length(p)^{1/2}}$$

5. Results & Analysis

5.1 Supervised Policy Learning Experiments

The results of the various data augmentation experiments done on supervised policy learning are shown in table 4. We observe that each of the data augmentation steps resulted in 2-3% average improvement in "Success %" measure for each task on the test dataset. This low gain is somewhat disappointing, it indicates that retraining with rewards is the crucial and having a good reward can improve accuracy drastically.

Task	Experiments			
	Baseline	**More intermediate nodes**	**More Training Samples**	**Longer training**
agentbelongstoorganization	0.44	0.44	**0.46**	0.45
athletehomestadium	0.22	0.14	0.22	0.14
athleteplaysforteam	0.14	**0.22**	**0.16**	0.08
athleteplaysinleague	0.28	0.26	0.18	**0.34**
athleteplayssport	0.18	**0.24**	0.16	0.12
organizationheadquarteredincity	0.58	0.58	0.58	**0.6**
organizationhiredperson	0.20	0.18	**0.28**	0.24
personborninlocation	0.52	0.50	**0.58**	0.58
personleadsorganization	0.44	0.36	0.44	0.32
teamplaysinleague	0.06	**0.34**	0.12	0.34
teamplayssport	0.01	0.04	0.04	**0.26**
worksfor	0.26	**0.30**	0.28	**0.34**

Table 4: Success % for Supervised Policy Learning

5.2 Retraining with Rewards Experiment - Modified Reward Factor Function

We chose the task 'athlete-plays-in-league' for this experiment, due to the large number of reasoning paths (7) discovered by the baseline model and a lack of a dominant path. Our results, as shown in table 5, indicate that increasing the penalization does not significantly improve the accuracy (MAP) while decreasing the penalization negatively impacts the accuracy. Our hypothesis is that, given that we use the

weight learnt for the original reward factor for the modified reward factor, the original reward factor already seems to severely penalizes the longer paths, the modified reward factor, $r_{efficiency+}$ has insignificant impact on the accuracy.

	$r_{efficiency}$ **(Baseline)**	$r_{efficiency+}$	$r_{efficiency-}$
No. of paths	7	6	7
RL MAP	0.5271	0.5297	0.5116

Table 5: MAP for athlete-plays-in-league Task

5.3 Error Analysis

Despite numerous challenges in implementation, we were able to almost reproduce the baseline results from the paper. Given the large number of relation tasks, shown below is an extract of the performance for the "*works_for*" relation.

Method	**Mean Avg. Precision**
TransE	0.23211373641279612
TransR	0.28138092408160065
TransH	0.2528149703511917
TransD	0.2389054998240636
RL	**0.48180613236416103**

Table 6: MAP for works_for Task

Given below are examples of correct and incorrect fact prediction by the RL method for the *works_for* relation:

Fact (e_{source} , e_{target} , relation = *works_for*)	**Correct**	**Hypothesis**
jerry_yang, website_yahoo_mail	Y	
ceo_vinod_khosla, company_sun	Y	
ceo_jim_balsillie, research_in_motion	Y	
vinod_khosla, company_sun_micro_systems	N	Long textual patterns on entity names in general seem to result in poor performance because of sparsity. Compositional approaches for modeling text [7] may help.
gerard_arpey, musicartist_american	N	Probably wrong relationship due to the closeness of American with American Airlines, where Gerard Arpey was a CEO.
donald_graham, company_tnt	N	The reward function might have influenced this path to encourage diversity. Donald Graham doesn't have any relation with TNT.

Table 5: Qualitative Analysis

We also make the following observations for the NELL-995 dataset on ***FactPrediction*** task

- The setup of the model doesn't seem to exploit the presence of inverse relations fully and in many instances comes up with reasoning paths with inverse relations that are not sound. For example, for the relation task, "person-born-in-city", the model derives the reasoning path:
  `concept:persongraduatedfromuniversity -> concept:persongraduatedschool_inv -> concept:personbornincity`

- The training distribution of relations doesn't seem to reflect that of test set

6. Conclusion and Future Work

In this work, we investigated the DeepPath model for KG reasoning. We evaluated the model against the NELL-995 dataset and analyzed the results. DeepPath is a complex system to be improved upon in a short span of time. The original implementation has several issues - functional bugs and extremely poor training speeds due to inefficient implementation. A significant portion of our time was spent in fixing these bugs, leaving us less time for experimentation. Nevertheless, this exercise helped us gain a perspective of the difficulties in implementing an RL-based system. Our main motivation behind choosing this project was to carefully study the key ideas (such as Monte Carlo, SARSA, Q-learning, DQN etc.) in reinforcement learning as applied to NLP, especially from the perspective of recent evolutions in neural architectures and to develop intuitions behind them and we believe we have achieved that learning goal, even though we could not improve upon the method presented in DeepPath. As a future work, we plan to carry on this study further, including the following aspects

1. Investigate different neural network architectures for the policy network used by the agent, including LSTMs to exploit the sequential nature of the path finding problem, similar to [4].
2. Investigate the technique on two other reasoning tasks. Namely, Knowledge Base Completion (mh-KBC) [4] and Path Query Answering (mh-PQA) [5]

We greatly appreciate our mentor, Horace Chu, for providing constant encouragement and valuable feedback through this work. We would also like to thank the authors of the DeepPath paper for releasing the original implementation [16] for us to build upon.

References

[1] DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning, Wenhan Xiong, Thien Hoang and William Yang Wang, Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017

[2] Ni Lao, Tom M. Mitchell, and William W. Cohen. 2011b. Random walk inference and learning in a large scale knowledge base. In EMNLP, pages 529– 539. ACL.

[3] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, David Gondek., Distant Supervision for Relation Extraction with an Incomplete Knowledge Base, Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies

[4] Rajarshi Das, Arvind Neelakantan, David Belanger, Andrew McCallum, Chains of Reasoning over Entities, Relations, and Text using Recurrent Neural Networks, Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers

[5] Traversing Knowledge Graphs in Vector Space, Kelvin Guu, John Miller, Percy Liang, 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)

[6] Translating Embeddings for Modeling Multi-relational Data, Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Advances in Neural Information Processing Systems 26 (NIPS 2013)

[7] Representing Text for Joint Embedding of Text and Knowledge Bases, Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, Michael Gamon, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing

[8] One-Shot Relational Learning for Knowledge Graphs, Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, William Yang Wang, Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing

[9] Random walk inference and learning in a large scale knowledge base, Ni Lao, Tom Mitchell, William W. Cohen, Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing

[10] A shortest paths ranking algorithm, José Augusto de Azevedo , Joaquim João E. R. Silvestre Madeira , Ernesto Q. Vieira Martins , Filipe Manuel A. Pires

[11] Toward an architecture for never-ending language learning, Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, Jr., Tom M. Mitchell, AAAI'10 Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence

[12] Knowledge graph embedding by translating on hyperplanes, ang, Zhen and Zhang, Jianwen and Feng, Jianlin and Chen, Zheng, Twenty-Eighth AAAI conference on artificial intelligence

[13] Ronald J Williams. 1992. Simple statistical gradient- following algorithms for connectionist reinforce- ment learning. *Machine learning*, 8(3-4):229–256

[14] Matt Gardner, Partha Pratim Talukdar, Bryan Kisiel, and Tom M Mitchell. 2013. Improving learning and inference in a large knowledge-base using latent syntactic cues. In EMNLP, pages 833–838.

[15] Arvind Neelakantan, Benjamin Roth, and Andrew Mc-Callum. 2015. Compositional vector space models for knowledge base completion.arXiv preprintarXiv:1504.06662.

[16] https://github.com/xwhan/DeepPath